ELM-ART: An Adaptive Versatile System for Web-based Instruction

Gerhard Weber, Peter Brusilovsky

▶ **To cite this version:**

Gerhard Weber, Peter Brusilovsky. ELM-ART: An Adaptive Versatile System for Web-based Instruction. International Journal of Artificial Intelligence in Education, 2001, 12, pp.351-384. hal-00197328

HAL Id: hal-00197328

https://telearn.hal.science/hal-00197328

Submitted on 14 Dec 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

International Journal of Artificial Intelligence in Education (2001), 12, 351-384

ELM-ART: An Adaptive Versatile System for Web-based Instruction

Gerhard Weber[1] and **Peter Brusilovsky**[2]
[1]*Department of Psychology, University of Education Freiburg D-79117 Freiburg, Germany,*
[2]*School of Information Sciences, University of Pittsburgh, Pittsburgh, PA 15260, USA,*
e-mail: webergeh@ph-freiburg.de, plb@sis.pitt.edu

Abstract: This paper discusses the problems of developing versatile adaptive and intelligent learning systems that can be used in the context of practical Web-based education. We argue that versatility is an important feature of successful Web-based education systems. We introduce ELM-ART, an intelligent interactive educational system to support learning programming in LISP. ELM-ART provides all learning material online in the form of an adaptive interactive textbook. Using a combination of an overlay model and an episodic student model, ELM-ART provides adaptive navigation support, course sequencing, individualized diagnosis of student solutions, and example-based problem-solving support. Results of an empirical study show different effects of these techniques on different types of users during the first lessons of the programming course. ELM-ART demonstrates how some interactive and adaptive educational components can be implemented in WWW context and how multiple components can be naturally integrated together in a single system.

INTRODUCTION

The maturity of the Web presented both a new opportunity and a challenge for researchers and developers working on all kinds of computer-supported educational applications. Naturally, it has also inspired a number of researchers in the area of adaptive and intelligent educational systems. Started in1995 from early efforts to move existing intelligent tutoring systems (ITS) technologies to the Web (Brusilovsky, 1995; Nakabayashi et al., 1995) a small stream of research on Web-based ITS has grown up into a steady broad current. Dozens of adaptive Web-based educational systems have been reported at the time of writing this paper. A recent review (Brusilovsky, 1999) has shown that all well-known technologies from the areas of ITS and adaptive hypermedia (AH) have already been re-implemented for the Web. The context of Web-based education has inspired a number of new ideas in the areas of ITS and AH and even lead to the development of some new exciting adaptive technologies. Overall, the marriage between adaptive educational systems (AES) and Web-based education was very creative for the AES.

It was expected that this marriage would also be creative for another side -Web-based education as an application area. AES researchers believed that an ability to adapt is more important for Web-based educational application than for all kinds of pre-Web systems. A number of reasons were cited in AES literature. Here we want to mention just two that we consider most important. First, most Web-based educational applications are used by a much wider variety of users than any standalone application. A Web application designed with a particular class of users in mind may not suit other users. Second, students usually work with Web-based educational systems on their own (often from home) and can't get an intelligent and personalized assistance that a teacher or a peer student can provide in a normal classroom. However, regardless of many possible reasons for Web-based education to employ adaptive and educational technologies, adaptive systems are not used in regular Web-based education. Instead, almost all schools and colleges involved in Web-based education rely on *courseware management systems* (CMS) - a new kind of systems emerged during the last few years. There are dozens of CMS available from various companies, consortia, and universities. Most popular

CMS such as WebCT (WebCT, 1999) or Blackboard CourseInfo (Blackboard, 2000) are used by hundred and thousands educational providers. These systems have zero adaptivity but they provide something that their customers appreciate most - versatility.

It turned out that versatility is more important for both teachers and administrators than any specific advance feature. Just a few years ago most Web-based educational systems were specialized. Some systems supported discussion forums, others supported Web quizzes, yet others let the teacher to place their course notes on the Web (Brusilovsky & Miller, 2001). The recent market-driven competition clearly showed that most users prefer a more versatile systems to a less versatile even if the latter has some advanced features. Modern CMS that has successfully survived this competition are able to support virtually any routine function of a teacher in the classroom. Let us consider, for example, a programming course developed with Blackboard CourseInfo. This system will support the teacher to provide a hyperlinked course material to read, programming examples to analyze, quizzes to take and programming problems to solve. Moreover, it will auto-grade all Web quizzes and will help the student to submit program solution to the teacher for grading. It will also let the student monitor their progress (in the form of grades) and communicate with the teacher and each other using e-mail, forums, and a chat room. Yes, the system is completely static, but it does support most of the teacher's real needs without an overhead to install, support, and learn more than one system.

Versatility is the aspect where adaptive and intelligent Web-based educational systems (AIWBES) lag very much behind CMS. As we can see from the review cited above (Brusilovsky, 1999), AIWBES can support virtually every function offered by modern CMS much better than a CMS due to the extra value provided by adaptive and intelligent technologies (See table 3). For example, it can provide adaptive navigation support for browsing the course notes (Brusilovsky, Eklund & Schwarz, 1998), adaptive generation of questions in quizzes (Rios et al., 1999), or adaptive peer help in discussion forums (Greer et al., 1998). However, each existing AIWBES can rarely support more than one function and it can hardly promote the use of these systems in practical Web-based education.

We do not claim that the lack of versatility is the only problem that prevents AIWBES from being used in Web-based education, but we do think that it is a major problem and that more efforts should be devoted to the development of versatile AIWBES. A versatile system is more appealing for practical Web-based courses. It also has chances to provide better adaptation since various components of this system can collect more information about student's knowledge, preferences, and interests. We consider it as a challenge to AIED community to develop versatile AIWBES with as many components as possible employing adaptive and intelligent technologies.

To provide an example of a successful versatile AIWBES this paper presents ELM-ART, one of the first and currently most comprehensive AIWBES. Versatility was always one of the driving forces in ELM-ART development due to the practical nature of the system. Starting with the first version developed in 1996 (Brusilovsky, Schwarz & Weber, 1996a) and over the following years the system was used for teaching real university course based on LISP programming language. Version by version, the needs of practical Web-based education have forced the development team to add some important features missed in the earlier versions. The current paper summarizes briefly the history of ELM-ART development and then presents in detail the most recent version of the system. Since the focus of the paper is versatility, our goal was not to concentrate on the single technology or aspect of ELM-ART, but to present all essential features of the system in context. We start with an overview of the system as a learner sees it (Section 3) and then present the full dissection of the system (Sections 4-9). For the features that are based on adaptive and intelligent technologies we provide a comprehensive description covering both the functionality and the underlying knowledge organization. For the features that are not yet adaptive we augment a brief description with some discussion on their possible adaptive implementation. Finally, Section 10 presents some empirical evaluations of ELM-ART. This section sheds some light on the value of adaptivity in the WWW context. In conclusion we discuss the role of WWW in the advancement of ITS research, the prospects of bringing ITS into practical Web-based education, and the role of versatile systems in this process.

THE HISTORY OF ELM-ART

The WWW-based introductory LISP course ELM-ART (ELM Adaptive Remote Tutor) is based on ELM-PE (Brusilovsky & Weber, 1996; Weber & Möllenberg, 1995), an on-site intelligent learning environment that supports example-based programming, intelligent analysis of problem solutions, and advanced testing and debugging facilities. The intelligent features of ELM-PE are based on the ELM model (Weber, 1996b). For several years, ELM-PE was used in introductory LISP courses at the University of Trier. The course materials were presented to students in regular classes (along with printed materials) as well as to single students working on their own with the printed materials only. Students used ELM-PE to practice the new knowledge by working on exercises. In this way, they were able to acquire the necessary programming skills.

ELM-PE was limited by the platform-dependent implementation of the user interface, the large size of the application, and the requirement for powerful computers available at universities only. These limitations hindered a wider distribution and usage of the system. So, we decided to build a WWW-based version of ELM-PE. The first step was to translate the text of the printed materials into WWW-readable form (html files), dividing it into small subsections and text pages that are associated with concepts to be learned. These concepts were related to each other by describing the concepts‘ prerequisites and outcomes, building up a conceptual network. When presenting text pages in the WWW browser, links shown in section and subsection pages and in the overview were annotated corresponding to a simple traffic lights metaphor referring to information from the individual learner model (Schwarz, Brusilovsky & Weber, 1996).

The second step was to port ELM-PE functionality to the Web. It enabled the very first version ELM-ART to provide live examples and intelligent diagnoses of problem solutions. When the learner clicked on such a live example link, the evaluation of the function call was shown in an evaluator window similar to a listener in ordinary LISP environments. Users could type solutions to a programming problem into an editable window and then send it to the server.

The approach of converting printed textbooks to electronic textbooks used in the first version of ELM-ART has been developed further in InterBook (Brusilovsky, Schwarz & Weber, 1996b), an authoring tool for creating electronic textbooks with adaptive annotation of links. However, from our first experiences with using ELM-ART, we understood that printed textbooks are not suitable for being transformed to hypertext pages in electronic textbooks in a one-to-one manner. Textbooks are usually written in sequential order so that single pages cannot be read easily when they are accessed from any page within the course. Additionally, the simple adaptive annotation technique used in ELM-ART had to be improved. Users should get more information about the state of different concepts that they had already visited and learned or had to learn. And, perhaps most importantly, inferring the knowledge state of a particular user from only visiting (and possibly reading) a new page is not appropriate (as correctly pointed out by Eklund, 1996). These objections and shortcomings were the motivation for enhancing ELM-ART.

In ELM-ART II (Weber & Specht, 1997), exercises and tests were added to the system. Results of working at these exercises and tests allowed the system to assess the student’s knowledge more carefully and to infer the user’s knowledge state. The student model was enhanced in a way that the annotation of links informed learners of whether they successfully worked at a page, whether the system inferred (form more complex or advanced knowledge units) that the learner already possessed the knowledge to be learned on a page, and whether the users already had visited the page. Additionally, communication tools were added. First, learners could send messages or questions to tutors. Tutors were informed by email and could respond via email, too, or could send a message to the user’s student model so that the messages were displayed when they entered the next page. Second, learners could correspond directly with other learners via a chat room.

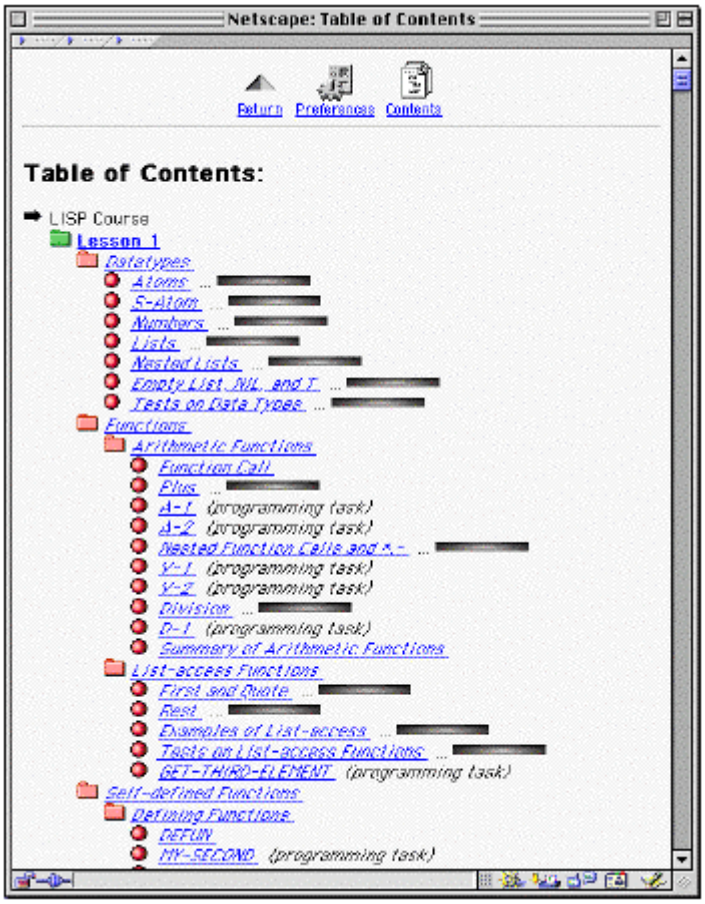

Figure 1. Part of the table of contents of the LISP-Course ELM-ART.

There are some other AIWBES inspired by ELM-ART II. First, the statistics tutor AST (Specht et al., 1997) used exercises and tests, too. The information system on drugs, ADI (Schöch, Specht & Weber, 1998), was done by the same authors as AST and added another feature, the inspectable and editable user model. The feature was added and enhanced in the next version of ELM-ART. And a first attempt was made to create an authoring system that supports building adaptive web-based courses. It resulted in the ACE-system (Specht & Oppermann, 1998).

In the next version of ELM-ART, the multi-layered overlay model was introduced and enhanced (Weber, 1999). Besides the knowledge states described above, now users were able to declare knowledge units as already known. All information on the user's knowledge state was held in parallel by the system so users could change their user model whenever they wanted or switch back to the original state without any loss of information. And two new communication tools were added. First, users can enter a discussion list where they can post messages that can be read by all other users of the course. The discussion lists are maintained by tutors. Second, users can communicate in a user group and can post and fetch documents among members of the group they belong to. However, this feature only works for users that are pre-registered by a tutor.

This latest version of ELM-ART was the basis of the new authoring system NetCoach (http://www.net-coach.de). With NetCoach (Weber, Kuhl & Weibelzahl, 2001), authors can create adaptive web-based courses that are based on the multi-layered overlay model, that support different types of test items, and that include all the communication tools mentioned above. NetCoach can be used via the web and offers templates to describe and link pages, to add exercises and test items, to adjust the interface and to set parameters that influence different features of the courses. With NetCoach, authors can create fully adaptive and interactive web-based courses without being required to program or to learn a programming language.

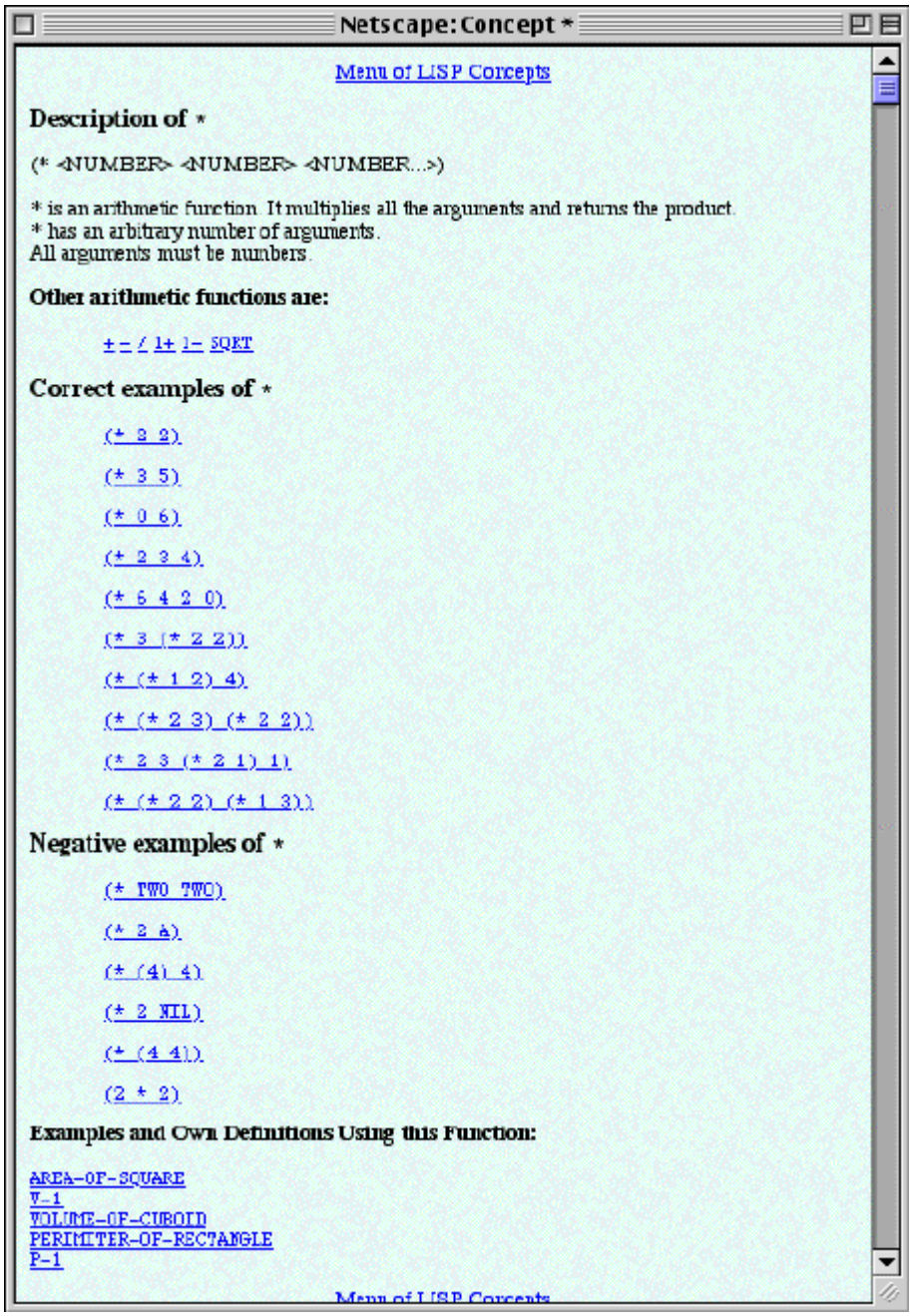

Figure 2. A fragment of ELM-ART on-line manual explaining the meaning of main navigation buttons and some toolbar buttons.

All versions of ELM-ART as well as all its direct descendant systems from InterBook to NetCoach were implemented with the programmable Common Lisp WWW server CL-HTTP (Mallery, 1994). Some implementation details can be found in (Brusilovsky et al., 1996a). ELM-ART can be accessed via the following URL: http://apsymac33.uni-trier.de:8080/Lisp-Course.

OVERVIEW OF ELM-ART

As we have mentioned above, ELM-ART has been originally designed as an integrated system supporting a range of features. We think that one of the most important problems in the design of an integrated system is the proper choice of a metaphor for the system organization. Current

CMS use a loose integration approach that we can name as "bag of features": different functionalities of the system are accessible using different items in a top-level menu. In contrast to this approach we have proposed a tight integration approach using an easy to understand metaphor of a textbook. A traditional programming textbook is a well-organized learning tool and our users did feel familiar with its features. A good programming textbook has a hierarchically structured content with a detailed table of contents and sections filled with textual presentation, figures, and programming examples. It also has a section with questions and programming exercises at the end of each section that presents new material. Best textbooks also include a glossary of all programming constructs with brief explanations. We have originally designed ELM-ART as a textbook of new generation that is built upon strength of the existing textbook model, but also adds new features that are unavailable in a printed textbook - interactivity and intelligence (Schwarz et al., 1996).

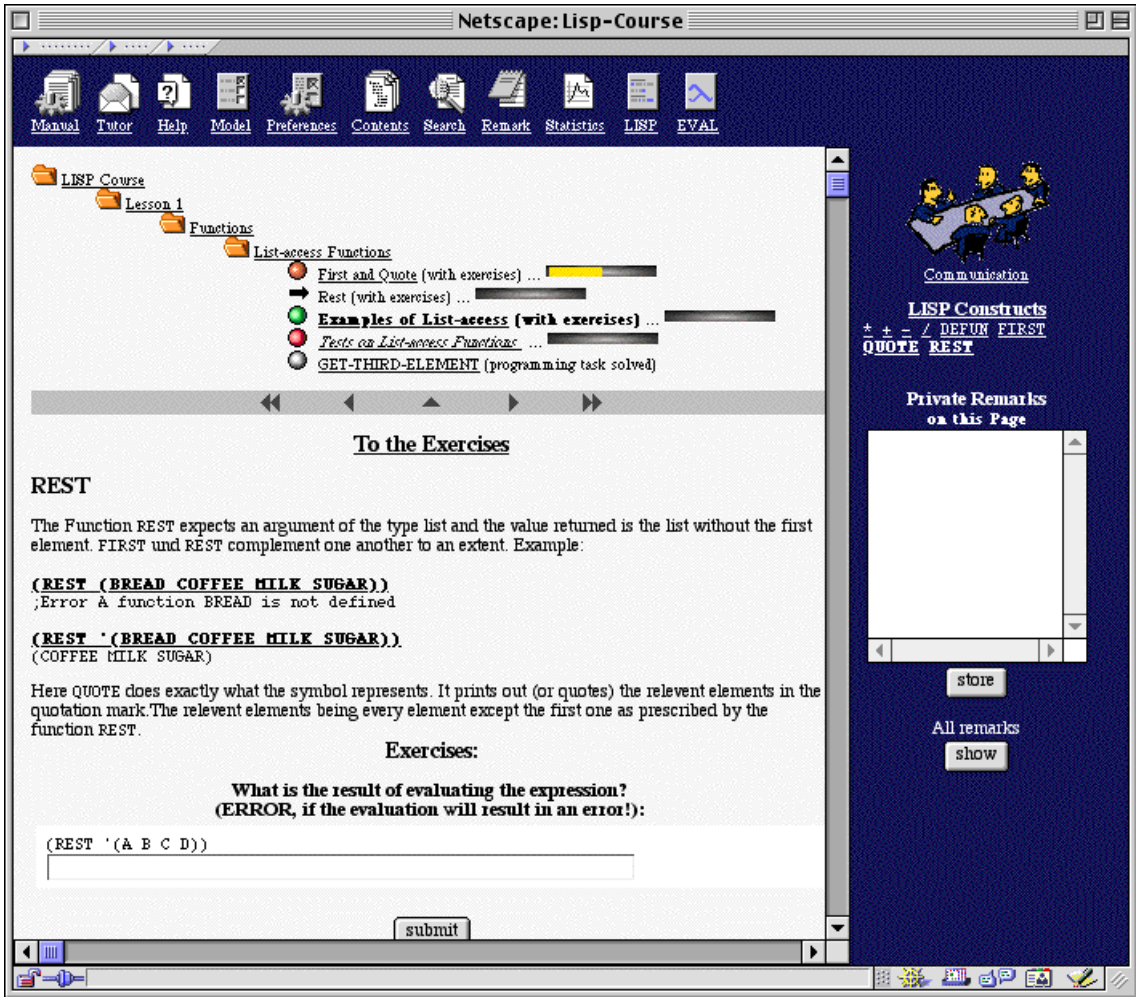

Figure 3. Example of a textbook page with an exercise. All LISP expressions are formatted as links providing one-click access to the evaluator code. The upper part of the window is the navigation center that shows the location of the page in the textbook and provides additional navigation choices. Colored folder and bullet icons provide and example of adaptive navigation support. Links on the sidebar connect this page to relevant glossary pages and example pages.

For a user of ELM-ART, the system can be viewed as an interactive intelligent textbook that is represented electronically in hypertext form. Like any programming textbook it is organized as a tree of sections from chapters on the top level to units on the bottom "leaf" level. Each section forms a separate page in the hyperspace. ELM-ART provides a detailed table of contents (ToC) that also serves as a navigation tool enabling users to get to any section of the book in one click (Figure 1). In addition to the tree of sections, the hyperspace also contains a

smaller tree for the glossary of constructs - one page for each LISP construct (Figure 2). ELM-ART supports several types of navigation including sequential (page-by-page) and hierarchical (parent-child), and glossary navigation. Each section of the textbook (Figure 3) contains the navigation bar with links to sequentially previous and next sections and the parent section (see explanation of navigation links on Figure 4). Above the section content the system provides a *navigation center* that clearly shows the position of the current page in the hierarchy and allows one-click navigation to all ancestor pages and to sibling pages on the same level. On the right sidebar of a section, ELM-ART generates links to all *glossary pages* and all *program examples* related to the current page. Thus from navigation point of view ELM-ART can be compared with the very best hierarchical electronic textbooks.

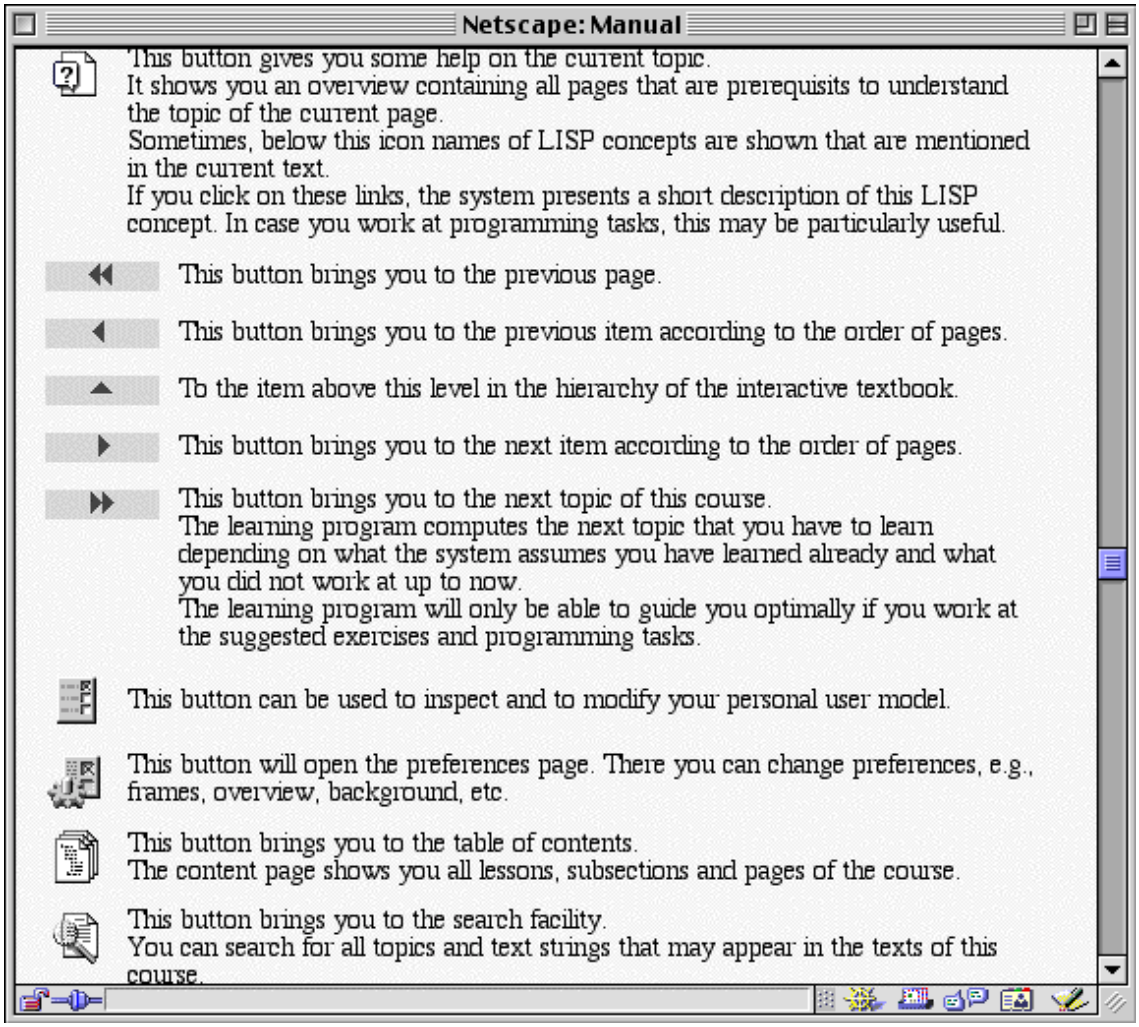

Figure 4. A fragment of the ELM-ART on-line manual explaining the meaning of the main navigation buttons and some toolbar buttons.

Unlike typical hypertext electronic books ELM-ART's textbook is not a tree of static pages. Its pages include several types of *interactive activities*. First, ELM-ART pages may include *live examples*. A live example is visible to users as an underlined LISP expression inside a book or a glossary page (Figures 2, 3, 8). A click on this expression brings a user to the interactive LISP evaluator (see Section 8) that immediately evaluates this expression and returns the result. The user can further explore this example using the evaluator by editing it and executing it again. In addition to simple evaluation mode, the evaluator includes a stepping mode that provides a visual step-by-step execution of the example (Figure 11). Another kind of live examples are problem-solving examples - a statement of a problem followed by a complete function definition that solves this problem (Figure 5). Users can explore the example solution

using the LISP evaluator by stepping through one of the sample function calls suggested in the problem statement or constructing own function calls.

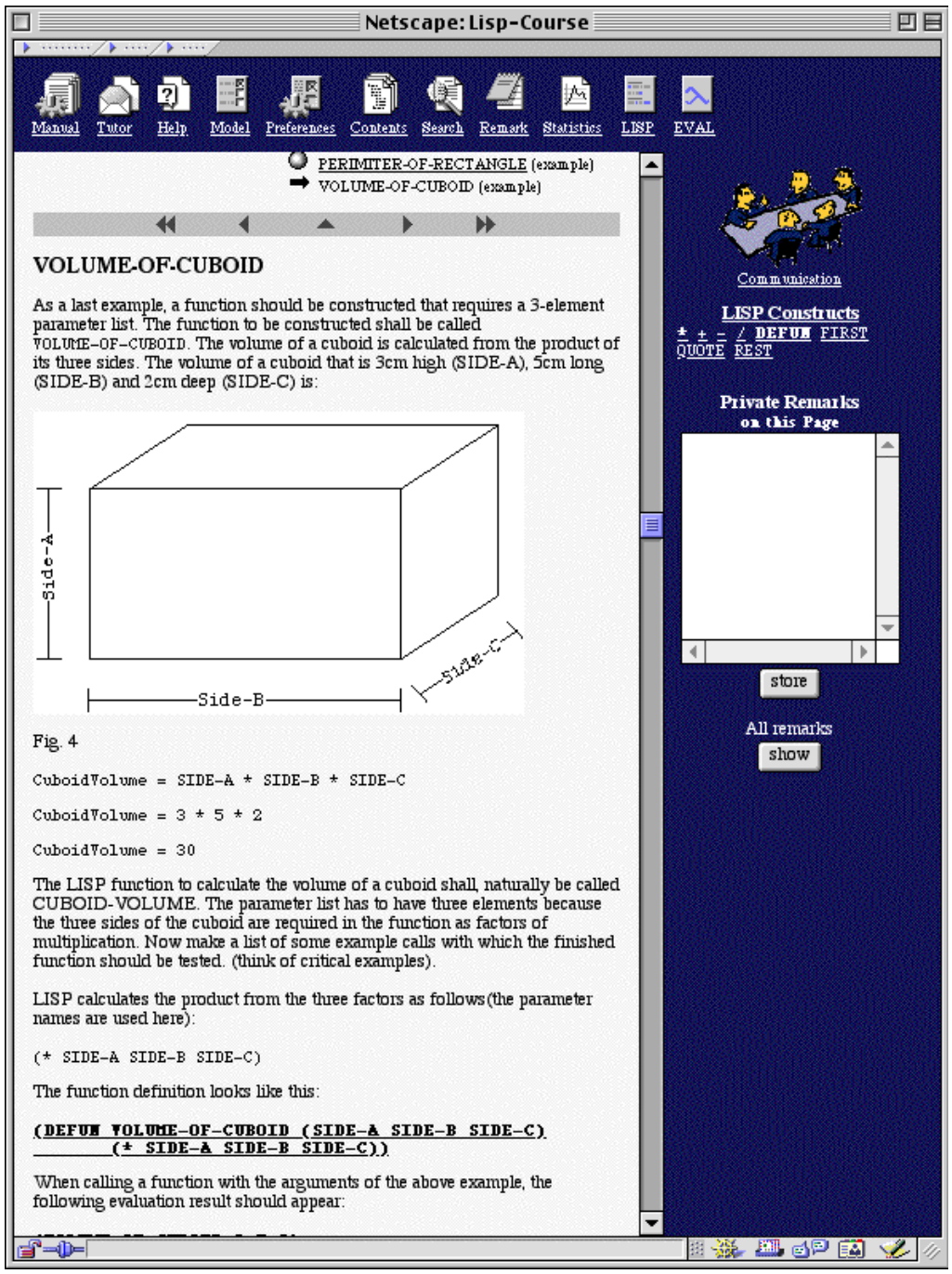

Figure 5. A page with an analysis of a problem-solving example. Using links to the evaluator students can explore the example program.

In addition to live examples, book pages may contain simple questions (Figure 3) and programming problems to solve (Figure 8). Users can test their knowledge by answering questions and solving programming problems. The system can automatically evaluate the correctness of answers and program solutions and provide immediate feedback. For programming problems, ELM-ART can also provide a detailed diagnosis of an incorrect solution (complete or incomplete) identifying student's errors and providing hints. Moreover, ELM-ART provides example-based problem solving support: the user who does not even know how to start the solution can ask ELM-ART about a reminder (a program example analyzed or

developed by this student in the past). The system is able to suggest up the most helpful reminder - the one that provides most help for solving the given problem. Naturally, the learner can extensively use the interactive evaluator when developing a solution for the programming problem. ELM-ART also provides some editing support by properly formatting student programs, To end the list of interactive features of the textbook, we should add that the user can add a personal remark (annotation) to every section of the textbook (Figure 3).

The electronic textbook in ELM-ART is also intelligent. It provides several kinds of support usually provided only by a human teacher. The most advanced features of the system are the already mentioned intelligent problem solving support and the intelligent reminder selection. These features are based on years or research and make ELM-ART unique among others Web-based educational systems in the area of programming. Another intelligent feature of ELM-ART is an ability to build adaptively the most relevant learning path for every learner. ELM-ART is always able to determine what is the best next learning activity for the user. Within a unit the system can challenge the learner with a sequence of questions that is adapted to the learner's current level of knowledge. For a unit-to-unit navigation, ELM-ART provides a special "next" link that is connected to the most relevant next unit for the given user (naturally, it can be different units for learners with different knowledge). In critical points ELM-ART can generate special navigation hints advising or discouraging the user to proceed with a particular page. For a learner who is interested to navigate between units on his or her own, ELM-ART provides adaptive navigation support in the form of colored bullets placed next of all links to units. The color of this bullet informs the learner of the educational status of the unit, for example, is it ready to be learned, does it contain any new knowledge. These adaptive visual cues help the learners to make an informed navigational choice at every point.

As one can see, ELM-ART's electronic textbook is very different from existing electronic textbooks in many aspects. The last aspect that we want to stress is that this textbook is not static - it is different for different learners. The system itself personalized the book to each user. On the same page the learners with different knowledge can get different questions, different navigation advises and different visual cues. The users also participate in making their books personalized. They can customize the book's look and feel (Figure 7). They fill book pages with personal remarks and unique problem solutions. As an additional support to this process the system enables users to view and navigate their personal "portfolio" - the set of all analyzed examples and all problem solutions. The system automatically structures this portfolio in a hyperspace of examples where each analyzed example of a solution is shown on a separate page (Figure 10). Each program analyzed or written by the student is automatically inserted into this portfolio. To connect this new page to the rest of the ELM-ART hyperspace, the system automatically generates bi-directional links from any new program to all similar programs and also to glossary pages of constructs used in this example (Figures 2 and 10). Finally, learners are even able to edit their "learner models" (Figure 6) thus implicitly changing navigation cues, sequence of examples, and other system-driven personalization.

ADAPTIVE ELECTRONIC TEXTBOOK

A Word about Knowledge in ELM-ART

According to Self (1995), different levels of knowledge have to be distinguished in intelligent learning environments. In his DORMORBILE framework, Self postulates that each agent (e.g., learner, teaching system) encompasses four knowledge levels. Learning processes in intelligent learning environments consist of interactions between agents at specific knowledge levels. The knowledge levels represent a hierarchy with higher levels including the capabilities of the lower levels. The four levels are domain knowledge, reasoning knowledge, meta-knowledge, and reflective knowledge, in increasing order. The ELM-ART framework mainly works on the lower two levels. Domain knowledge (we also refer to it as conceptual knowledge) consists of all the predicates, functions, and symbols an agent knows (asserts to be true) that are required to solve problems in the given domain. Domain knowledge is declarative and is assumed to be independent of any particular problem. The next higher level, the reasoning knowledge,

concerns procedural knowledge which uses domain knowledge to solve problems. These distinctions allow us to separate single units of knowledge from their use.

In ELM-ART, this distinction manifests itself in two different types of knowledge representation. On the one side, the electronic textbook with all the lessons, sections, and test units is founded mainly on domain knowledge and deals with acquiring this knowledge. On the other side, the episodic learner model ELM deals with procedural knowledge necessary to solve particular programming problems. Sections 4 and 5 describe the representation of the domain knowledge level and the functionality supported by this level of knowledge. The reasoning knowledge level with the ELM model and the supported functionality will be described in Section 6. Section 7 presents some features of the system that are supported by both domain and reasoning level of knowledge.

The Electronic Textbook: Knowledge Representation

ELM-ART represents knowledge about units to be learned with the electronic textbook in terms of a conceptual network (Brusilovsky et al., 1996a). Units are organized hierarchically into lessons, sections, subsections, and terminal pages (units). Terminal pages can introduce new concepts, present lists of test items to be worked at, offer problem-solving examples or suggest problems to be solved. Each unit is represented as an object containing slots for the text unit to be presented with the corresponding page and for information that can be used to relate units and concepts to each other. Static slots store information on prerequisite concepts, inference links, and outcomes of the unit (the concepts that the system assumes to be known when the user worked on that unit successfully). Units have a tests slot that may contain the description of a group of test items the learner has to perform (described in Section 5). When test items have been solved successfully the system can infer that the user possesses the knowledge about the concepts explained in this unit. Problem pages have a slot for storing a description of a programming problem.

The user model related to this declarative conceptual domain knowledge is represented as a multi-layered overlay model.

- *Visited State.* The first layer describes whether the user has already visited a page corresponding to a unit. This information is updated whenever the learner enters a page.
- *Learned State*. The second layer contains information on which exercises or test items related to this particular unit the user has worked at and whether he or she successfully worked at the test items up to a criterion or solved the programming problem.
- *Inferred State*. The third layer describes whether a unit could be inferred as known via inference links from more advanced units the user already worked at successfully. Whenever a unit has been recognized as learned, this information will be updated in all inferred units (via inference links). This is a recursive process that stops whenever a unit already has been marked as learned or inferred.
- *Known State*. The fourth layer describes whether a unit has been marked as already known. This can be done explicitly via the collaborative student modeling approach (see next section) or implicitly by selecting a specific learning goal (not used in the current version of ELM-ART).

Information in the different layers of the learner model is updated independently during each interaction with ELM-ART. So, information from each source does not override others. In Section 6, we describe another type of student model (ELM) that is used for individualizing problem solving support.

Collaborative Student Modeling for the Electronic Textbook

One objection to intelligent tutoring systems is the possible inadequacy of automatic student modeling. One can not be sure that the information a system has gathered is sufficient to model

the learners' knowledge and skills correctly. This may be especially true in the case of learning a programming language where students may have very different starting level of knowledge and speed of learning. It is not surprising that one of the first known ITS for programming BIP (Barr, Beard & Atkinson, 1976) let the students provide their own estimation of knowledge of the subject. For the same reason we have decided to include a tool for inspecting and modifying the student model into ELM-ART.

In ELM-ART a learner is able to inspect his or her current student model to see what the system actually assumes to be true about him or her and, possibly, to modify the model's assumptions. On each page of the course, the learner can select the model button in the toolbar frame at the top of the window. The current browser window displays information about the status of the current page and, in case the page is not a terminal page, information about all pages directly below this level will be shown (see Figure 6).

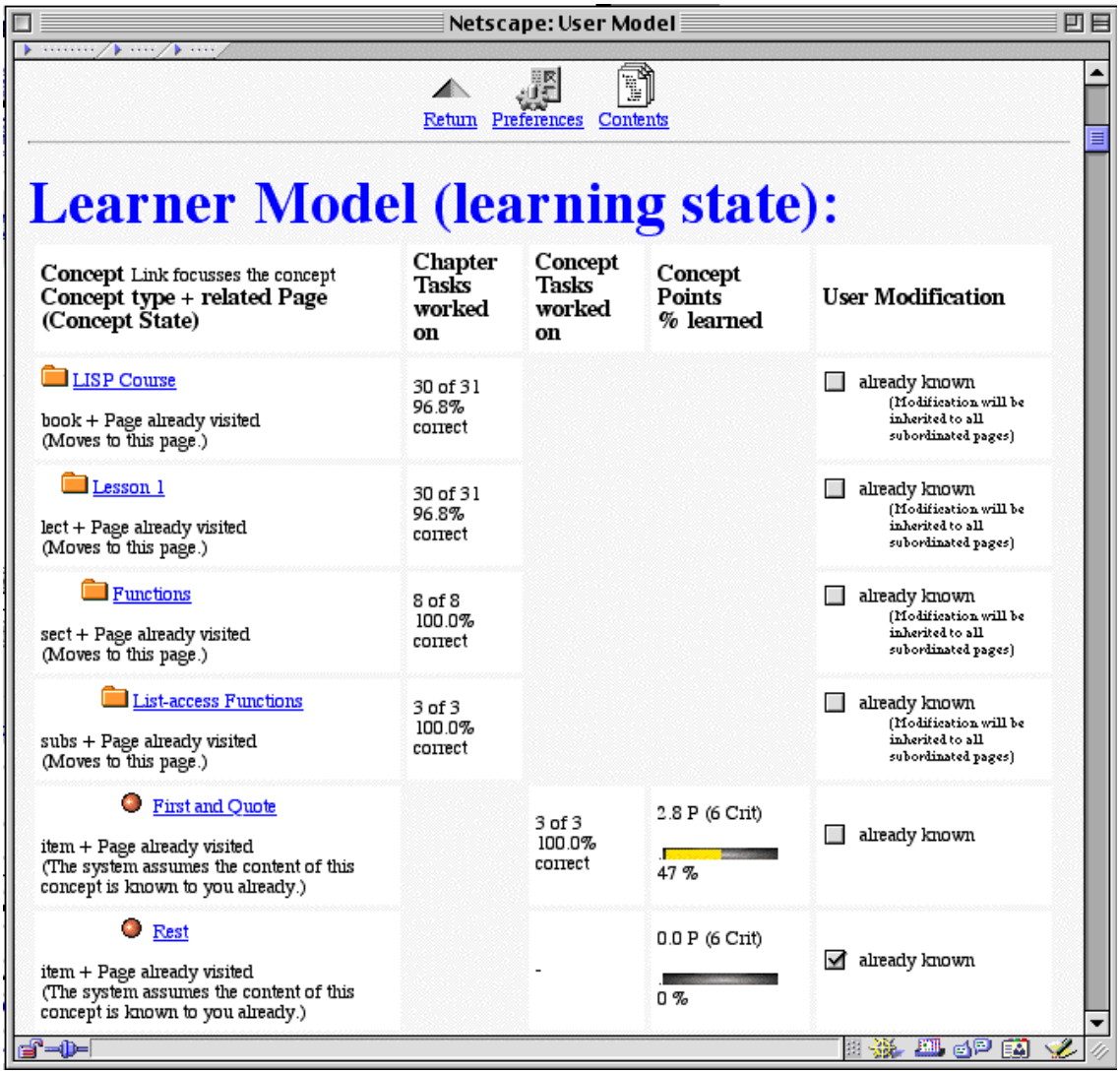

Figure 6. The open and editable student model in ELM-ART lets the students observe the state of the model and provide own estimation of their knowledge.

For each page, the system shows a table with columns for the name of the page with its current annotation, the explanation of the annotation, the learning status (in case of a page with exercises or tests), and, in the last column, the user's appreciation. In this last column, the user can mark that he or she already knows the content of this page and all subsequent pages. Similarly, the user can remove the mark. If a page is marked as known by the learner, the system will trust him or her and annotate the page accordingly. When guiding the user, the

system will not lead the learner to a page that he or she marked as already known. However, this information coexists with the system's knowledge about the user. If the user successfully worked at a unit's exercises or tests, the system will annotate the page as successfully solved. Additionally, if the user removes the mark from the page, the system's previous estimation will be used for annotating the page. With these multiple sources of information about the learning status of page from the multi-layered overlay model, no information will get lost and the user can revise the model back and forth at any time.

A viewable and editable student model in ELM-ART is not a complete innovation. Even before we have started to work on ELM-ART, this concept has been discussed and investigated by two groups of researchers in the UK and Australia (Bull & Pain, 1995; Kay, 1995). We hold, however, that in the context of a practical Web-based ITS this kind of student model is not a research issue, but a necessity. With the emerging importance of life-long education, WWW-based instructional systems will be used by learners who have some prior knowledge of the domain to refresh or extend their understanding. Other students may have experiences in related domains and, therefore, will have some initial understanding by analogy. Finally, with a greater variety of students even very good student modeling techniques may fail to maintain a precise student model. An open student model let the students and the system collaborate in the process of maintaining the most up-to-date information about student knowledge.

What is Adaptive in the Electronic Textbook

Visual adaptive annotation of links

The multi-layered overlay model supports adaptive annotation of links. Links that are shown in an overview on each page or in the table of contents are visually annotated according to the learning state of the corresponding unit. ELM-ART uses an extension of the traffic lights metaphor to annotate links visually (see Figures 1, 3, 7 and 8). Green, red, white, and orange balls are used to annotate the links (additionally, the texts of the links are outlined in different styles to aid color-blind users).

A *green* ball means that the system suggests visiting the page behind the annotated link and the concepts taught on this page are ready to be learned. That is, all prerequisites to this concept have been learned already or inferred to be known.

A *red* ball means that this page is not ready to be visited. In this case, the learner does not know at least one of the prerequisite concepts (that is, the system cannot infer from successfully solved tests and programming problems that the user will possess the required knowledge). However, ELM-ART does not restrict learner's navigation. A learner is allowed to visit this page and in the case that he or she solves the corresponding test or programming problem correctly, the system infers backwards that all of the necessary prerequisites are known that are declared by different inference relations. This is a very strong assumption in diagnosing the learner's knowledge state.

A *white* ball has different meanings depending on the type of the page the link points to. In the case of a terminal page with exercises, a test page, or a problem page, the white ball means that the exercises, tests, or the problem have been solved correctly. In the case of any other terminal page, the white ball indicates that this page has been visited already. In the case of a lesson, section, or subsection link, the white ball means that all subordinate pages have been learned or visited.

An *orange* ball has different meanings, too. In the case of a terminal page, an orange ball means that the system inferred from other successfully learned pages that the contents of this page are known to the learner (as described above). In the case of a lesson, section, or subsection link, an orange ball means that this page has been visited already but not all subordinate pages have been visited or solved successfully. If the user informed the system that he or she already knows the content of the section or the page, the link will be colored orange unless the system has enough evidence (from solving exercises, tests, or programming problems) that the user already mastered the concepts connected with this link (then the link is colored white).

In browsers supporting JavaScript, the different meanings of a state of a link are explained in the status line at the bottom of the window when the cursor is located over the link.

Additional information (the current learning state of test items) is shown with links to pages that have tests or exercises. A horizontal bar indicates the percentage of points (up to a criterion) already reached by solving test items related to this unit (Figures 1, 3 and 6).

Adaptive navigation support with colored balls was pioneered by ELM-ART in 1996 and since that used in many adaptive AIWBES (Brusilovsky et al., 1998; Henze & Nejdl, 2001; Schöch et al., 1998; Specht & Oppermann, 1998; Specht et al., 1997; Weber et al., 2001). Empirical studies have shown several benefits of this kind of adaptation (Brusilovsky & Eklund, 1998; Brusilovsky & Pesin, 1998). Note that the meaning of different colors may not be the same in different systems using this technology.

Individual Curriculum Sequencing.

While adaptive annotation of links is a powerful technique to aid learners when navigating through the pages of the course, some users may be confused about what the best next step should be to continue with the course. This may happen when the learner moves around in the hyperspace and loses orientation. In other cases, the learner may want to follow an optimal path through the curriculum in order to learn as fast and as completely as possible. To meet these needs, a NEXT TOPIC button in the navigation bar of the text pages allows the user to ask the system for the best next step, depending on the current knowledge state of the particular learner (see Figures 3, 5, and 8). In addition, at the end of each unit the system may present an explicit suggestion of what should be learned next.

The algorithm to select the best next step for a particular user works as follows: Starting from the current learning goal, the system recursively computes all prerequisites that are necessary to fulfill the goal. The first concept belonging to the set of prerequisites that is not learned or solved already will be selected and offered to the learner. The learner completes the course successfully when all prerequisites to the current goal are fulfilled and no further goal can be selected.

What is Adaptable in the Electronic Textbook

Besides the importance of adaptivity in computer-based instructional systems, adaptable features that allow the user to tailor the learning system to his or her needs are necessary to make the system user friendly (Oppermann, 1994). The special architecture of ELM-ART computes each page individually. This allows the learner to specify *preferences* that are stored with the individual student model. These preferences will be considered by the system when generating a new page. In the current version of ELM-ART, the learner can specify, for example, whether they want to work with frames, whether the background will be colored, whether an overview of the current hierarchy of concepts will be shown at the top of each page, whether this overview will show the current learning state of test items of the page, whether links should be annotated, whether the user wants to be guided automatically, whether a frame with a personal annotation for the current page will be shown, and whether the text and the interface will be presented in English or German (Figure 7).

Many of these preferences were suggested by users when asking the tutor for help or when giving feedback. This is one of the advantages of a server-based learning system. The behavior of the system can be changed directly without updating the applications at each site or producing a new CD-ROM.

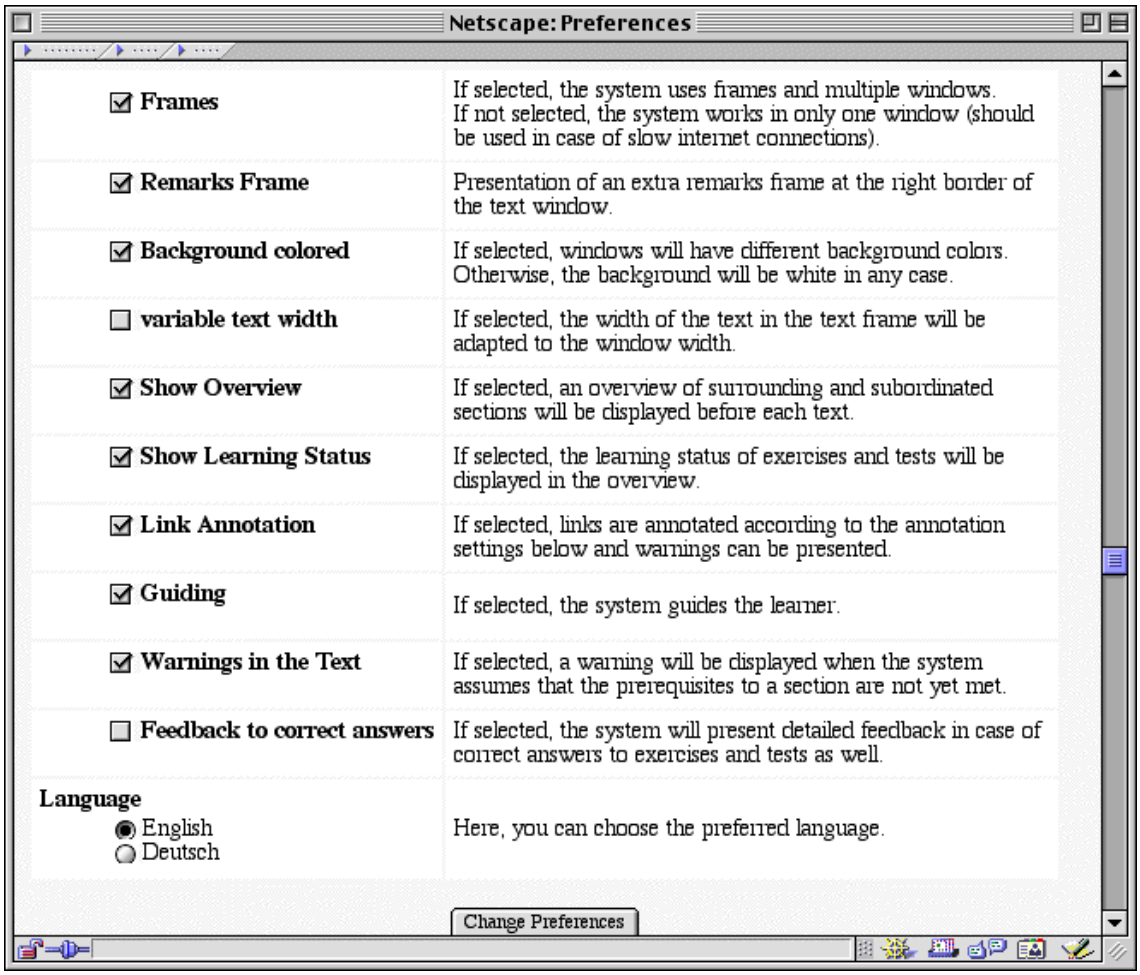

Figure 7. Students can adapt the look and feel of the electronic textbook to their individual preferences.

ADAPTIVE TESTING IN ELM-ART

Objective tests or quizzes is one of the most important features of modern Web-based educational systems (Brusilovsky & Miller, 2001). Especially in adaptive learning systems, testing has at least two means. First, testing is essential for an optimal learning process (Anderson, Conrad & Corbett, 1989) because it is accompanied with feedback on the correctness of the answers and with reasons why an answer is wrong. Immediate feedback allows the user to correct erroneous knowledge and misconceptions so that errors will not manifest in his or her knowledge and skills. Second, testing results are the most reliable source of evidence that a user has learned a concept. This information can be most trusted when computing the annotation of links and when computing the individual curriculum and guiding the learner to the next best page to be learned.

Test Items

ELM-ART was one of the first intelligent educational systems that has considered testing very seriously and included it into its core. It supports five different types of test items: yes-no test items, forced-choice test items, multiple-choice test items, free-form test items, and gap-filling test items. In yes-no test items users simply have to answer yes-no questions by clicking the “yes” or the “no” button. In forced-choice test items, users have to answer a question by selecting one of the alternative answers, and in multiple-choice test items users have to answer a question by selecting all correct answers provided by the system. In free-form test items,

users can type an answer to the question asked freely into a form. In gap-filling test items, users have to type in characters or numbers to complete a word or a sentence.

Test items are objects that contain information on the question text, the correct answers, the name of the function that checks the correctness of the user response in relation to the expected correct answers, a parameter that determines the difficulty of the test item, a slot with text describing a hint in case of an error, and a slot listing related concepts. The difficulty parameter determines how much evidence is added to the confidence value of related concepts when the test item is solved correctly. Test items are stored in a separate list and can be associated to test groups individually.

Test groups are collections of test items that are associated with a specific unit. Therefore, test items can occur in different test groups. If a test item has been solved on one page it will not be presented in a first run on another page (e.g., in a final quiz), but may be considered in test repetitions.

A unit will be considered as mastered when the confidence value of the unit has reached or exceeded the critical value of the unit. The confidence value of each unit can be set with as a parameter of the unit's test group. The algorithm of computing the confidence value is as follows. Each test item has a fixed difficulty value. In a test group, each test item that belongs to the test group is given a weight that will be multiplied with the test item's difficulty value. In case a test item is solved correctly, the product of the difficulty value and the weight is added to the unit's confidence value. In case the answer to a test item is wrong, the product of the difficulty value and the weight is multiplied by an error factor and then subtracted from the unit's confidence value.

Additionally, test items can influence concepts other than the unit of the current test group. Depending on whether a test item is solved correctly or not, the confidence values of all concepts listed in the related concepts slot of the test item are increased or decreased with a weighted proportion of the difficulty value, respectively. This mechanism gives the system the power of a very simple connectionist system.

In the LISP course, test items play a twofold role. On the one hand, test items are used to check whether the user possesses the correct declarative knowledge. This is especially useful in the beginning of the course when a lot of new concepts (data types and function definitions) are introduced. On the other hand, test items can be used in evaluation tasks to check whether users are able to evaluate LISP expressions correctly. Skills used in evaluation are the inverse of the skills used to generate function calls and function definitions. Evaluation skills are needed to decide whether programs work correctly and to find errors in programming code. Program creation skills are practiced in special tasks. They are supported by the episodic learner model approach described in Section 6.1.

Test groups

There are three different situations in which test items from test groups are presented to a user. First, test items can be used as exercises when learning a new concept. Second, test items can be used as a final test in extra test pages at the end of a section or lesson. And third, on a non-terminal page, test items selected from subordinate unit pages can be composed into an introductory test.

Exercises

When learning new concepts, test items can serve as exercises. This both helps the learner get feedback on his or her understanding of the new concept and provides the learning system with information about the knowledge state of the concept. This information is more valid compared to only inferring the learning state from simply visiting the page like in the first version of ELM-ART (Schwarz et al., 1996) or in InterBook (Brusilovsky et al., 1996b). The learner will be requested by the system to work at more exercise items as long as the confidence value does not reach the critical value. This is similar to knowledge tracing technology (Corbett & Anderson, 1992). The system starts with presenting one test item (Figure 3) from the test group with medium difficulty. In case of an error, the system will randomly select another test item

with lower difficulty, in case of no error, the system will randomly select two test items from the test group with higher difficulty.

The system gives feedback on the number of errors on the test items presented on the last page and presents all erroneous test items with both the users' answers and the correct answers. Next to the test item, a help button enables the user to access all pages that have information related to this test item. Additionally, an explanation is given as to why the answer provided by the system is the correct one.

Starting with the second exercise on a particular concept, exercise items are shown without the text of the unit page. The text of that page and all other related pages can be accessed by pushing a help button located near the test item. In this way, the learner is not overwhelmed by all the information and can easily access it if needed.

When the system has enough evidence that the unit is learned, no further exercises will be presented to the learner automatically. However, the learner can continue working at exercises by pushing the *more exercises* link displayed with the feedback to the last answers.

Final Tests

Test items can be used to wrap up a lesson, section, or subsection. The mechanism is similar to exercises in a unit page. However, a larger group of items (usually in the range of 6 to 10 items) is presented simultaneously, and there are no associated help buttons with each test item. More general help can be accessed via the help button from the button panel in the top frame of the window.

As with exercises, test items in final tests are linked to related concepts. So, solving final test items will lead to upgrading the confidence values of related concepts (or lowering it in case of errors). Therefore, working at a final test before working at the related concepts will implicitly influence the learning status of related concepts. When the final test is completed successfully, the system will infer that all prerequisites to these pages (as stated in the *prerequisites* slot) are known. Inferring a concept as being known to the learner does not have the same importance as the confidence value of the concept. If there is enough confidence that a concept is already learned, this information will be preferred. Only if the confidence value is below threshold will the inferred information be used. That is, indirectly inferring a concept as already known will not override the confidence information. The two kinds of information are stored in different layers in the learner model.

Introductory Tests

A user may be interested in showing the learning system that he or she already knows at least some parts of the topics to be learned in a lesson, section, or subsection. Similarly, he or she may want to see what type of questions have to be answered in the subsequent pages. To meet these needs, on a section level, a user can decide to start working at exercises before going deeper into subsequent pages by clicking on a link to a pretest section. The pretests will be gathered from the exercises from all subsequent unit pages. For practical reasons, the total number of items in a pretest is limited to about 25 test items with a maximum of 5 test items for each concept involved. The system assumes that a learner will be able to solve all the test items only if he or she already knows about the underlying concepts. For this reason, test items do not have to be practiced as long as exercises and the difficulty value is given higher weighting. If there are not too many concepts in the section, solving all test items correctly will raise the confidence value of most involved concepts above threshold, and the system will conclude that these concepts are already learned.

PROBLEM SOLVING SUPPORT WITH EPISODIC STUDENT MODELING

Episodic Student Modeling

The system's knowledge consists of both LISP domain knowledge (described above) and episodic knowledge about a particular learner. Both types of knowledge are highly interrelated. That is, on the one hand, the system is able to consider individual, episodic information for diagnosing code and for explaining errors in addition to using the common domain knowledge. On the other hand, when explaining individual errors and examples from the learner's individual learning history, the system can combine episodic information with information from the domain knowledge.

The representation of the domain knowledge used in episodic modeling consists of a heterarchy of concepts and rules (Weber, 1996a). Concepts comprise knowledge about the programming language LISP (concrete LISP procedures as well as superordinate semantic concepts) and schemata of common algorithmic and problem solving knowledge (e.g., recursion schemata). These concept frames contain information about plan transformations leading to semantically equivalent solutions and about rules describing different ways to solve the goal stated by this concept. Additionally, there are bug rules describing errors observed by other students or buggy derivations of LISP concepts which, e.g., may result from confusion between semantically similar concepts.

The individual learner model consists of a collection of episodes that are descriptions of how problems have been solved by a particular student. These descriptions are explanation structures (in the sense of explanation-based generalization, Mitchell, Keller & Kedar-Cabelli, 1986) of how a programming tasks has been solved by the student. That is, stored episodes contain all the information about which concepts and rules were needed to produce the program code the students offered as solutions to programming tasks. Episodes are not stored as a whole. They are distributed into snippets (Kolodner, 1993) with each snippet describing a concept and a rule that was used to solve a plan or sub-plan of the programming task. These snippets are stored as episodic instances with respect to the concepts of the domain knowledge. In this way, the individual episodic learner model is interrelated with the common domain knowledge.

To construct the learner model, the code produced by a learner is analyzed in terms of the domain knowledge on the one hand and a task description on the other hand. This cognitive diagnosis results in a derivation tree of concepts and rules the learner might have used to solve the problem. These concepts and rules are instantiations of units from the knowledge base. The episodic learner model is made up of these instantiations. In ELM, only examples from the course materials are pre-analyzed and the resulting explanation structures are stored in the individual case-based learner model. Elements from the explanation structures are stored with respect to their corresponding concepts from the domain knowledge base, so cases are distributed in terms of instances of concepts. These individual cases—or parts of them—are used by ELM-ART to provide two important kinds of problem-solving support. First, episodic instances can be used during further analyses as shortcuts if the actual code and plan match corresponding patterns in episodic instances. It enables the system to diagnose incorrect and even incomplete student programs. The ELM model and the diagnosis of program code is described in more detail in (Weber, 1996a). Second, cases can be used by the analogical component to show similar examples and problems for reminding purposes (Weber, 1996b). This is the technology behind ELM-ART's example-based programming feature.

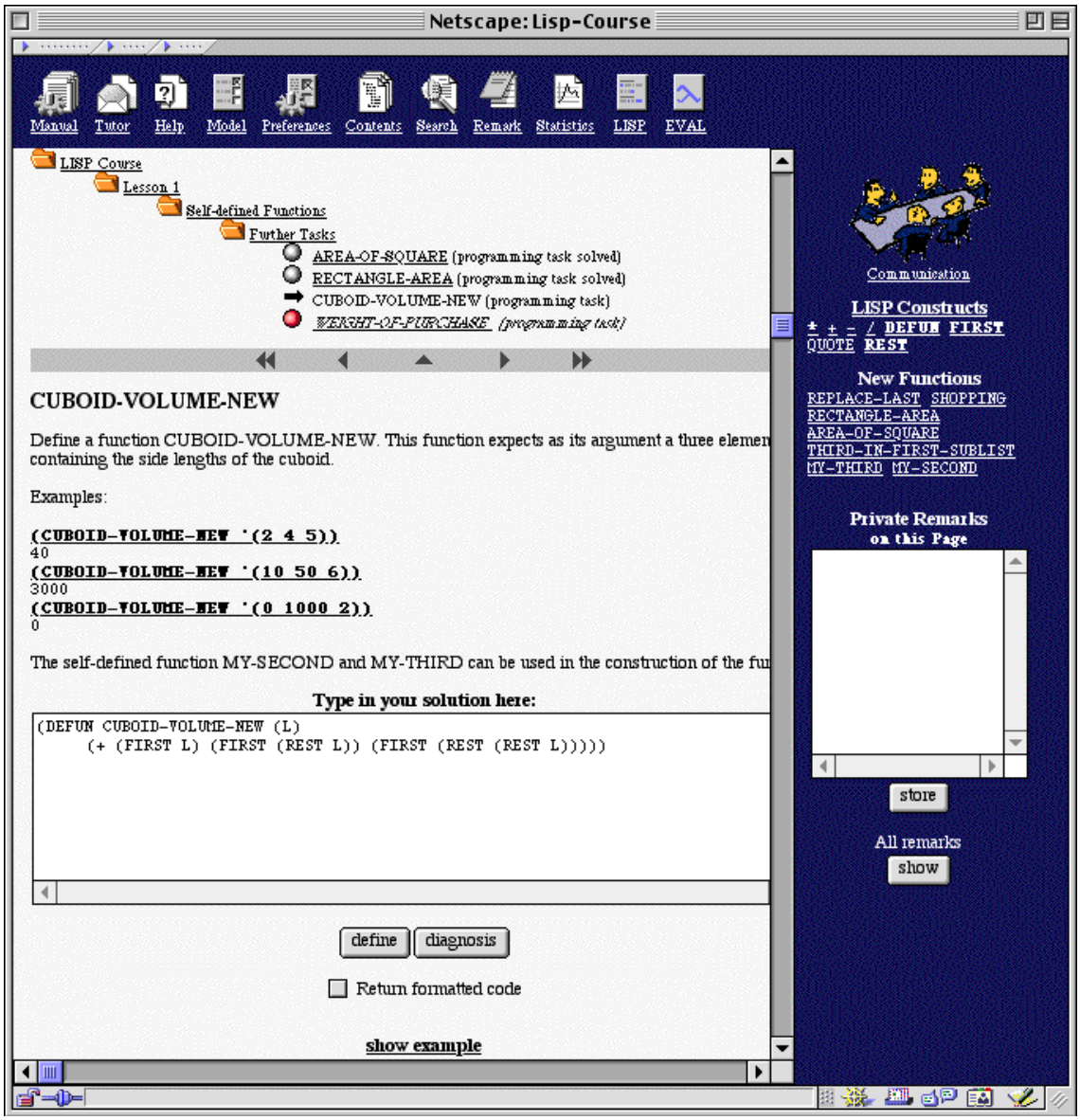

Figure 8. A page with a programming problem. The upper part of the window shows the problem location in the course. Links on the right sidebar offer one-click navigation to learned LISP concepts and to earlier explored or constructed functions. Buttons and links on the bottom offer problem-solving help.

Diagnosing and Testing Function Definitions

Programming in LISP mainly consists of coding function calls and defining new functions. ELM-ART has more than fifty such programming problems at its disposal. Each programming problem is presented on a separate page (see Figure 8). For function definitions, learners are encouraged to test each function definition on their own. After defining the function, the learner can select an example call of this function with typical arguments. These function calls (together with the expected result of the function call) are displayed in a list below the input area. The learner can simply select a function call. The evaluator window opens and shows the result of the function call (this is discussed in more detail in Section 8.4.1). In case of an error or if the result is not as expected, the learner should try to find the error on his/her own.

If the student failed to complete the solution to the problem, or if the student cannot find an error that was reported when evaluating the code in the evaluator window, he or she can ask the system to diagnose the code of the solution. The system gives feedback by providing a

sequence of help messages with increasingly detailed explanation of the error or suboptimal solution (Figure 9). The sequence starts with a very vague hint on what is wrong and ends with a code-level suggestion of how to correct the error or how to complete the solution. In many cases, the student can understand from the very first messages where the error is or what can be the next step and does not need any more explanations. The solution can be corrected or completed, checked again, and so forth. The student can use this kind of help as many times as required to solve the problem correctly. In this context, the option to provide the code-level suggestion is a very important feature of ELM-ART as a distance learning system. It ensures that all students will ultimately solve the problem without the assistance of a human teacher.

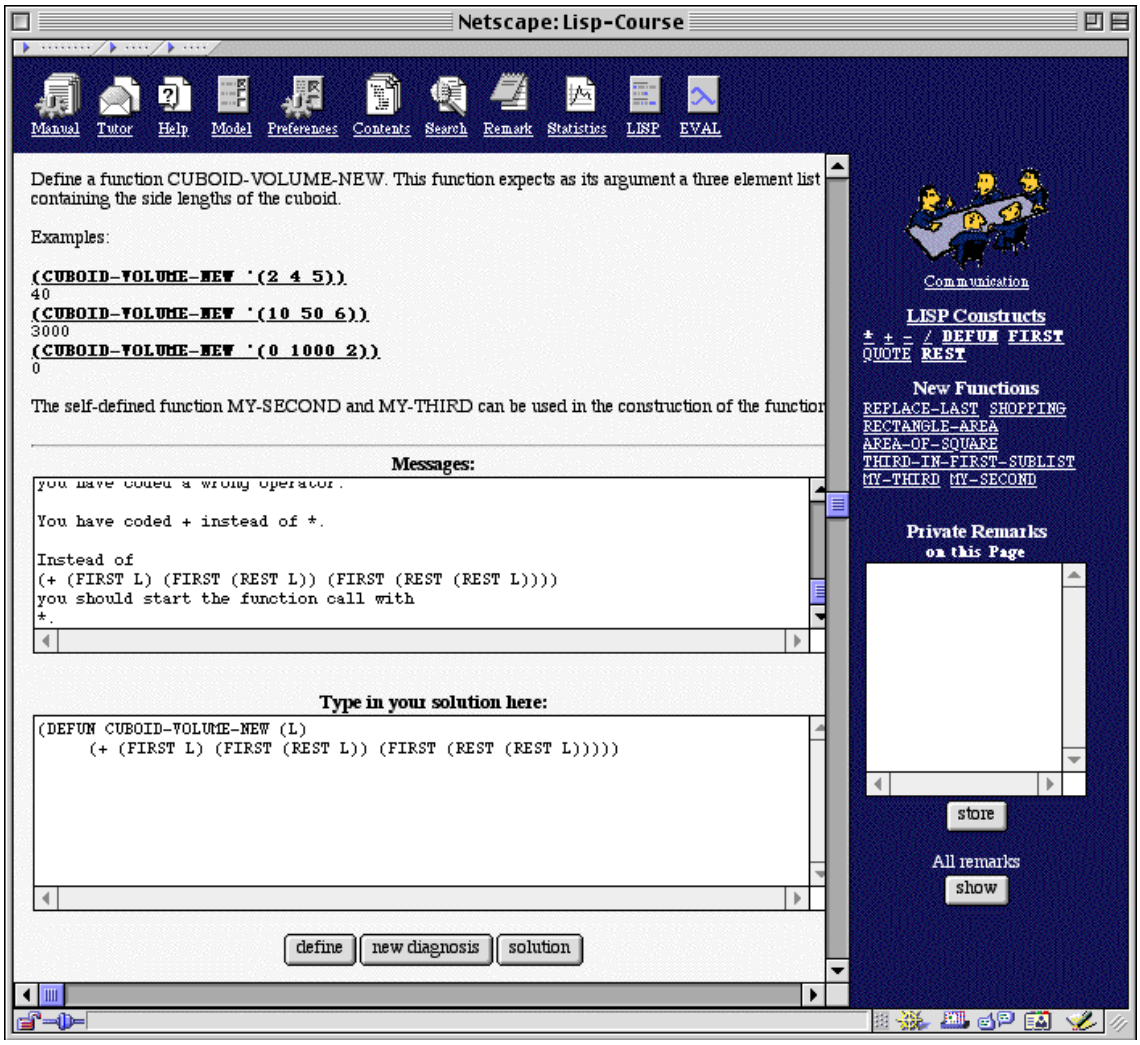

Figure 9. A diagnosis of an incorrect solution. The student can ask the system to diagnose his or her solution by pressing the "Diagnose" button on a programming problem page.

Example-based Programming

ELM-ART supports example-based programming. That is, it encourages students to re-use the code of previously analyzed examples when solving a new problem. The hypermedia form of the course and, especially, similarity links between examples help the learner to find relevant examples from his or her previous experience. As an important feature, ELM-ART can predict the student‘s way of solving a particular problem and find the most relevant example from the individual learning history. This kind of problem solving support is very important for students who have problems with finding relevant examples. Answering the help request, ELM-ART selects the most helpful examples, sorts them corresponding to their relevance, and presents

them to the student as an ordered list of hypertext links. The most relevant example is always presented first, but, if the student is not happy with this example for some reason, he or she can try the second and the following suggested examples. The implementation of this feature was adopted directly from ELM-PE (Brusilovsky & Weber, 1996; Burow & Weber, 1996). Additionally, users can navigate the hyperspace of examples to find the most relevant example. Both issues will be presented in more detail in the following sections.

Computing an Ordered List of Relevant Examples

To support the student selecting the proper example, ELM-ART is able to generate an ordered list of relevant examples. This list is generated by the Explanation-based Retrieval (EBR) algorithm which is described in detail in (Weber, 1996b). At the first step, the system uses its knowledge about the problem and the individual ELM to generate the most probable solution for the given problem. On the basis of concepts and rules that will be used to generate the new solution, the case memory can be probed for cases most similar to this solution. Concepts from the resulting explanation structure are inserted temporarily into the existing concept hierarchy of the episodic learner model. All episodic frames that are neighbors to the temporarily inserted frames contribute to computing weights for similar episodes. Competition among episodes is introduced by normalizing episodic weights with respect to the sum of weights of all organizationally similar neighbor frames. For each episode, the system computes the similarity value by summing up the resulting weights for all of its constituent episodic frames.

Finally, the system selects the episodes with the highest similarity values and presents a list of links to examples and reminders ordered according to their similarity values to the student. The most relevant example is always first in the list so the students who rely on the system's choice can easily use it. However, with this interface they have also a possibility to try the "second best" and other examples in the list. To give students more information for selecting an example, the system shows the similarity value for each example in the list.

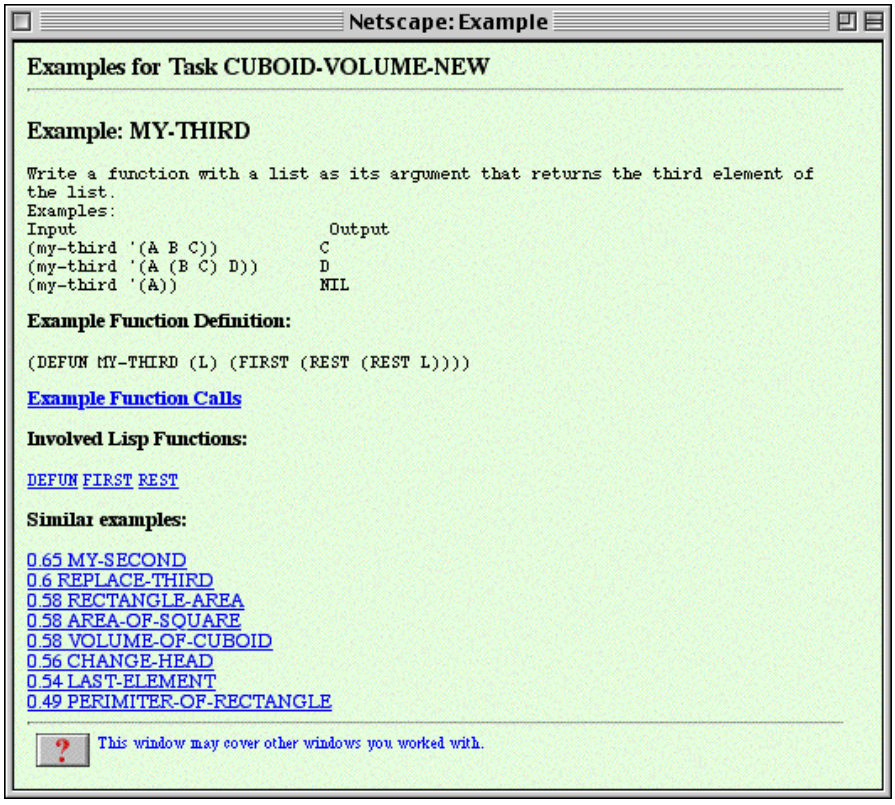

Figure 10. Example-based programming in ELM-ART. A window with the most relevant example from the student's own past experience is popped up after the student clicked the "Example" link on a programming problem page.

Similarity-Based Navigation Between Examples

Another level of support for selecting the proper example is provided by the hyperspace of LISP functions and examples described in section 7. Starting from the ordered list of examples suggested by ELM-ART or starting from the hypermedia manual (if the goal is to find an example related to a particular LISP-function) the student can explore this hyperspace with semantic and similarity links. The most interesting feature here is the similarity link. Some hypermedia-based information retrieval systems are able to generate similarity links between stored items of information (Tudhope, Taylor & Benyon-Davies, 1995). Experience with these systems shows that similarity-based navigation is a very powerful tool for searching in the hyperspace. In ELM-ART, the similarity links between examples are computed using a variant of the EBR-algorithm. As we mentioned above, this algorithm can generate the list of structurally similar example episodes and compute the similarity value for each of them. This list is presented to the student as a list of links from the given example to related examples. For each related example, the similarity value is shown and the list is ordered according to these values (see Figure 10). This interface provides the student with navigation support in similarity-based navigation. The navigation support in ELM-ART is adaptive because the similarity values are computed using the individual ELM.

THE HYPERSPACE OF LISP FUNCTIONS AND EXAMPLE DESCRIPTIONS

The hyperspace of learning material in ELM-ART is not limited to a human-authored electronic textbook. In addition to the hierarchy of units and pages described above, ELM-ART lets the student browse a hyperspace of LISP functions and code examples. The set of examples includes all function definitions that the students has created when solving a problem or explored as a problem-solving example. The set of functions includes all LISP functions used in the course. Each page of this generated hyperspace presents one object (a function or an example) providing a description of this object and links to related other objects (Figures 2 and 10). Unlike human-authored course pages, the pages in the hyperspace of functions and examples are completely generated from the system's internal knowledge about functions and examples. For each type of object we have simply provided a presentation function that extracts various slots from a frame-like description of an object and place them in predefined position in an HTML presentation pattern. In ELM-ART the content of a function description is not adaptive (Figure 2). However, we want to stress that computer-generated glossaries are generally a good object for adaptation. Recent research on adaptive text generation in electronic encyclopedias (Bontcheva, 2001; Milosavljevic & Oberlander, 1998) has presented a number of very useful adaptation techniques.

The LISP function pages form a subset of this hyperspace with a predefined structure. The goal of these pages is to provide a small LISP reference manual with a hypermedia interface. Links between LISP functions correspond to semantic relations between them (such as belonging to the same class of functions). The example part is completely dynamic. New example pages are added as soon as a student explores a new example or solves a new problem. Links from a newly added example to other examples and functions are established on the fly. An example is connected by bi-directional links to all functions, which are used in this example. This means that the student can navigate from a function to any existing example, which uses this function and backwards, from an example to all functions used in it. In addition, each example is connected to other examples by adaptive similarity-based links described above.

A user has several ways to jump into the hyperspace of functions and examples. First of all, the LISP reference manual with a list of links to all represented LISP functions is available for users at any moment. Second, in a problem-solving situation, the user can get a list of links to examples that are relevant to the task at hands ("show example" link on Figure 8). Third, course pages provide links to functions and examples that are relevant in its context (right sidebar on figures 3, 5, and 8). Starting from any of these points, the user can follow numerous semantic links to explore all the hyperspace of functions and examples.

Following a link to a LISP function brings the student to the function glossary window (Figure 2). This window shows a short description of the concept's meaning and syntax, a list of positive and negative examples, links to related concepts, and links to examples of function definitions (including self-defined functions) that use this LISP concept. Following a link to an example brings the student to the same example window that is used for example-based programming (Figure 10). All LISP functions used in the example and similar examples are shown as links. Following such a link, the learner will get to the description of a concept or another example as described above.

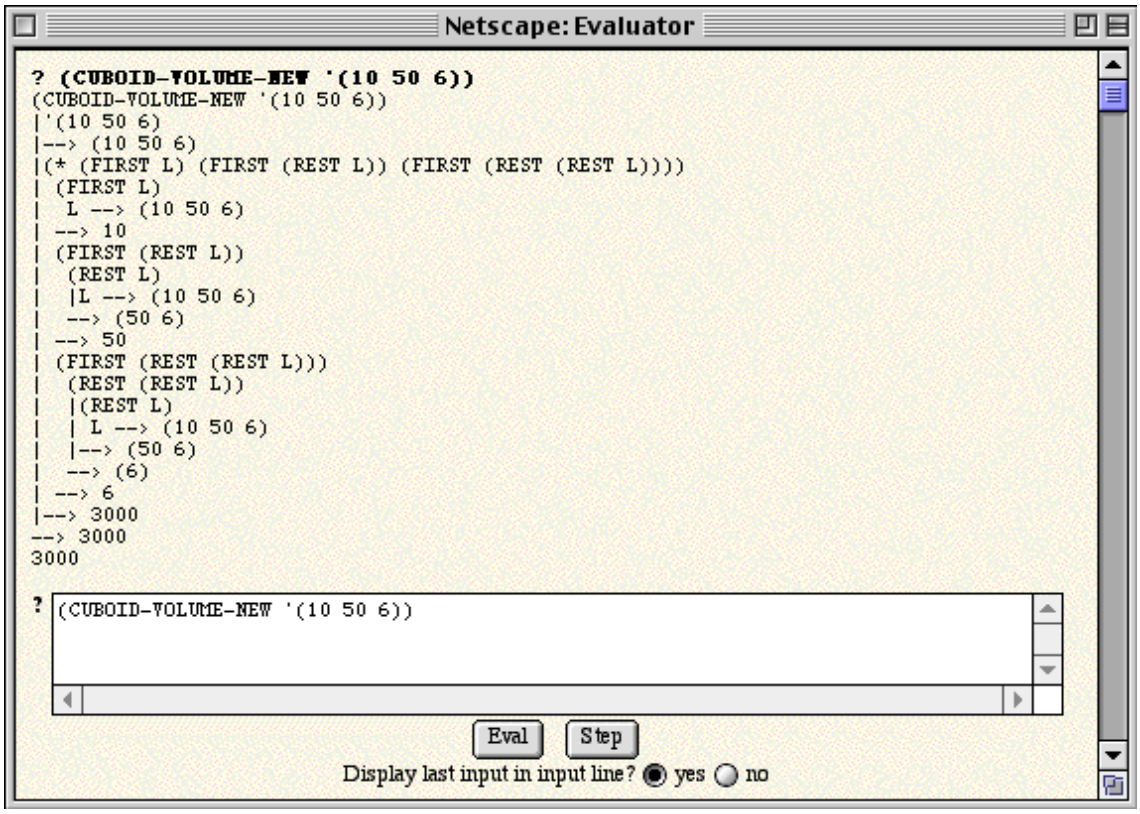

Figure 11. The interactive LISP evaluator can provide both the result of evaluation of a LISP expression and a step-by-step visualization of the evaluation process.

INTERACTIVE LISP EVALUATOR

ELM-ART combines the features of guidance by an intelligent tutoring system and of support by an open learning environment. In an open learning environment, a learner should be able to engage into exploration of domain concepts on his or her own. ELM-ART is an open learning environment in the way that it provides an interactive LISP evaluator. Learners can explore working in LISP by evaluating LISP expressions. The evaluator window can be accessed by pushing the EVAL button from the toolbar frame or by clicking on a "live" code example within a text page. In the evaluator window, LISP expressions can be typed directly into an input field (see Figure 11). The user can decide whether to simply evaluate the expression or to evaluate the expression in stepper mode. In stepper mode, the evaluator not only displays the result of the evaluation but also all evaluation steps performed to evaluate the code. This stepper mode is especially useful for understanding how LISP evaluates expressions and for finding run time errors. In case of an error, not only the error message is displayed but also an explanation of why this error may have happened. These explanations are especially useful for beginners, who can be confused by the usual error messages.

The server application simulates the evaluation of LISP expressions. It only supports functions and concepts needed in the programming course. With this simulation of LISP (within LISP) it is possible simply to catch run-time errors, endless loops, and non-terminating recursions. Additionally, a user will not be able to program the server because there is no direct access to the complete LISP environment the server (CL-HTTP server) is running in.

In ELM-ART an interactive evaluator is an important, but non-adaptive component. However, we can clearly see a natural way to make it adaptive using the same learner's model. The evaluator can simply take into account the strident knowledge level of lisp functions and constructs and provide adaptive levels of details in visualization. In brief, the constructs that are not learned yet can be visualized in detail, while the constructs that are already well learned - with no detail. This approach was successfully used in the past in ITEM/IP-II system (Brusilovsky, 1994).

COMMUNICATION TOOLS

Computer-based learning systems usually lack two important aspect of a learning situation: direct interaction with a human teacher or tutor and discussion with other learners. Modern CMS compensate the lack of face-to-face communication by including various communication tools such as E-mail, discussion forums, chat sessions, and video conferencing. We added some similar tools to ELM-ART to support most necessary human-to-human communication in the otherwise completely computer-based learning system. The communication page can be entered via the communication button.

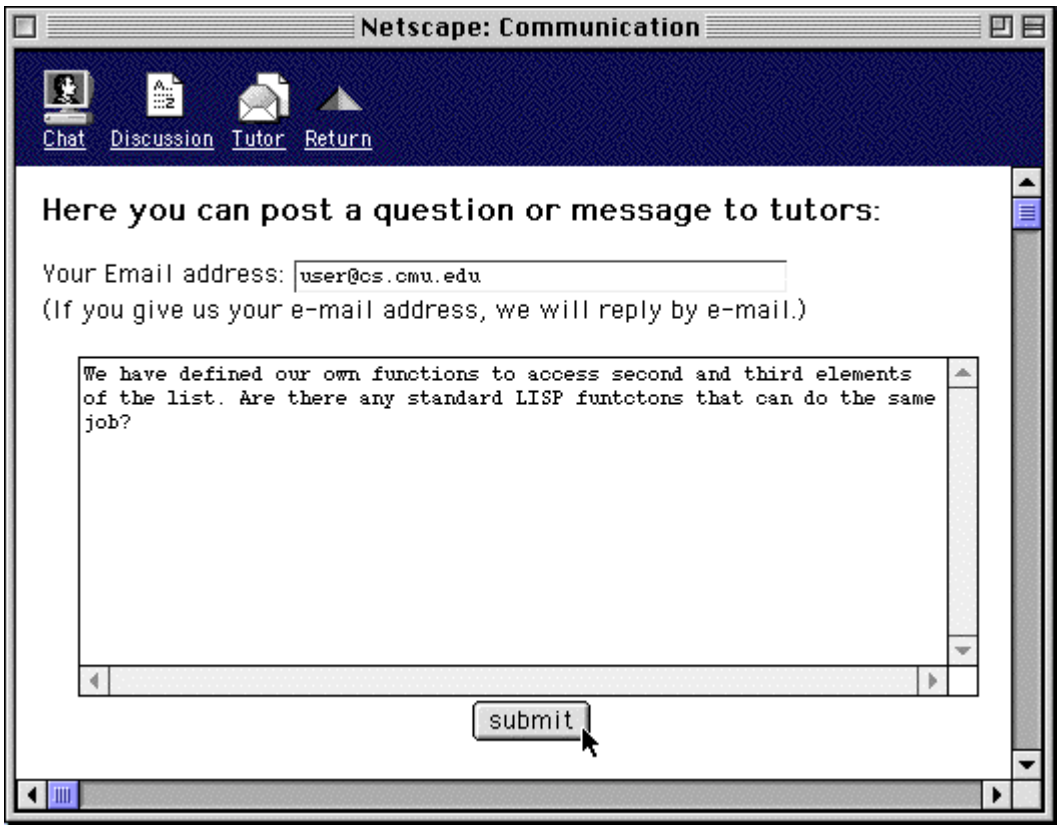

Figure 12. Communication with a tutor.

Communication with a Human Tutor

At any moment during working with ELM-ART, users can send a message to the human tutors supporting the learning course via the communication tool. Usually, the communication page opens with the tutor interface (Figure 12). In case another communication tool is displayed, the tutor interface can be accessed by clicking the “Tutor” button from the toolbar (Figures 3 and 12). Optionally, the learner can add an e-mail address (in case he or she did not provide the address when first logging in to ELM-ART). The server stores the message in the individual student model and directly sends e-mail with the messages to the tutors.

The tutors can react in two ways: by sending a message to the student model or by sending the message directly to the user via e-mail. Sending a message to the student model is very simple and works even if the user did not provide an e-mail address. When the user enters a new page while working with the learning course, the system checks whether the learner's record contains a message to the user. If there is one, a link is displayed at the top and at the end of the text frame. Following this link, the user will get into the tutor window. The editor window displays the message and then offers the possibility to respond to the tutors (see Fig. 8). Additionally, the user can mark the message as already read so the message will not be displayed again.

If the learner provided an e-mail address, the tutor can respond directly to the learner via e-mail. The advantage of this method is that the learner will be informed as soon as the tutor responded to his or her question, even if the learner is not currently working with ELM-ART.

Our experience with this communication tool shows that learners are using the message system to ask questions as well as to give feedback to the developers of the system. Most learners preferred to provide their e-mail address, however, some learners preferred to keep their anonymity.

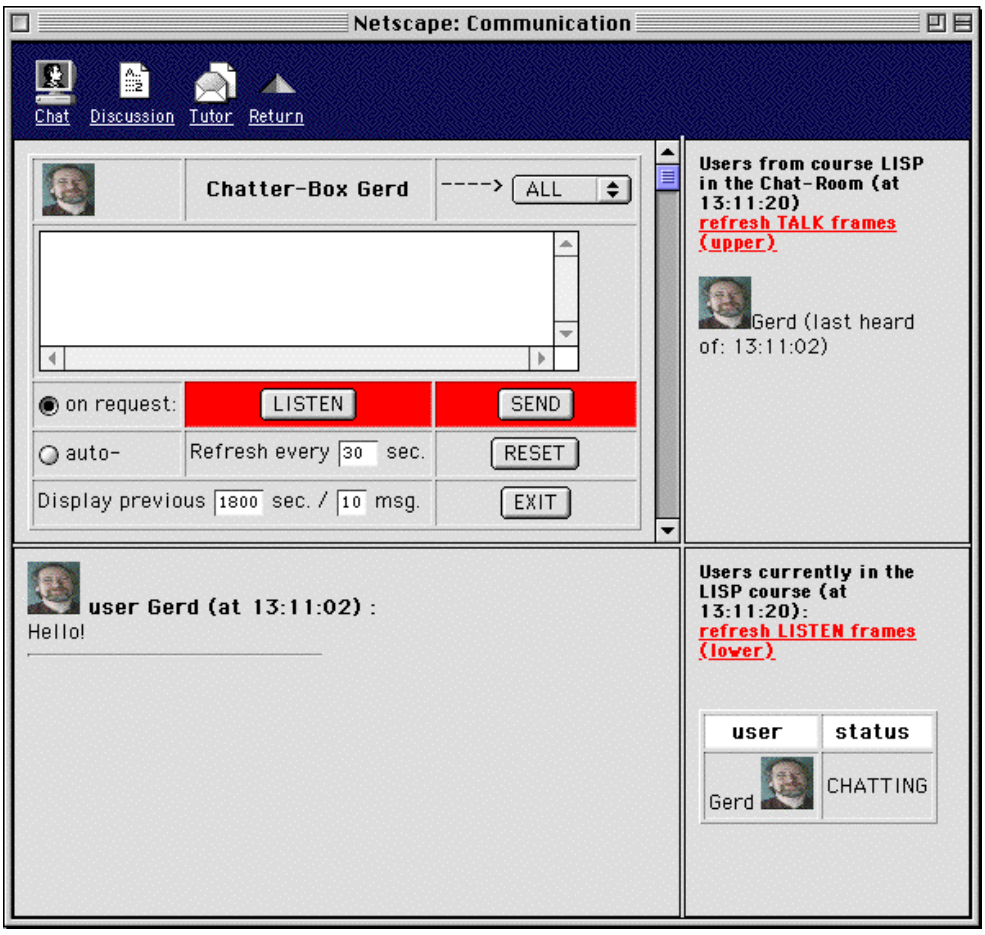

Figure 13. Communication in the Chat Room.

Chatting

ELM-ART's chat room is no different from many other chat rooms used in Web-based educational systems. Learners can get into the chat room via the communication link. First, the user gets an introductory dialog window where he or she can select an icon that will be displayed together with his or her name. Alternatively, the user can provide the URL to a picture (e.g., from his or her personal home page). This will make correspondence in the chat room livelier. Additionally, a short description of how to use the chat room is provided. In the next step, the user enters the chat room. There, icons or pictures of all users currently visiting the chat room are displayed along with the most recent messages. The user can type in a message into an input window and send it to the server (see Figure 13). At any time, the user can redisplay all current messages or ask the system to redisplay the most recent message automatically.

The chat room is particularly useful if some learners arrange to meet in the chat room in advance. Participants in the chat room can decide to send a message directly to a particular user (called whispering). When whispering, only the specified user will receive the message and no other user will see it. In this way, a type of privacy can be achieved even in the public chat room. From our own first experiences with using the chat room we learned that it is used very often for some type of social communication similar to private talks in traditional classroom situations. However, these are impressions only, because discussions in the chat room are not stored permanently or recorded in any way. Each message is erased after one hour.

Users are urged to exit the chat room actively. Messages from users that are no longer actively in the chat room are displayed with a special icon. If a user has not interacted with the server for longer than half an hour, the server will remove that user from the chat room.

Discussion lists

Chatting is only useful if learners want to discuss on the fly problems with other users or want to get into some social contact with users at distant places. However, some discussion may be useful for other learners and may even be relevant at later times. Therefore, a discussion list (also known as a discussion forum) is available in the current version of ELM-ART. Users can enter the discussion list via the communication link.

Users can add new contributions to different topics in the discussion list or respond to contributions directly. Tutors of the course can edit the discussion list. They can add new topics to the discussion list or can change already existing contributions to topics.

Towards adaptive communication support

As already mentioned, communication facilities are implemented in ELM-ART on the minimal level and without any kind of adaptation. In this aspect ELM-ART is no different from the majority of modern CMS. However, recent research on adaptive WBES has shown several ways to make discussion groups and collaboration over the distance intelligent and adaptive.
The group from the University of Saskatchevan has extended their original workplace-oriented peer-help technology to the WBE context in their Intelligent Helpdesk system (Greer et al., 1998). Another similar system was developed and evaluated at the University of Central Florida (Morelos-Borja, 1998). In addition to that, a group at the University of Duisburg known for their pioneering work on adaptive collaboration support (Hoppe, 1995) have recently suggested a complete framework for implementation of intelligent support techniques for distributed internet-based education. In all these cases, an ability to use and match the models of multiple learners connected to a system is the key of adaptive collaboration support.

EXPERIENCES WITH ELM-ART

Adaptive Annotation and Curriculum Sequencing

With the introduction of the second version of ELM-ART, we started an accompanying empirical study to look at the effects of combining the new adaptive annotation technique used in ELM-ART with the guidance offered by the NEXT button. As this study is reported in another paper already (Weber & Specht, 1997), we only will present the most important results here.

In a first study, two treatments with two levels each were investigated simultaneously. The first treatment contrasted the adaptive annotation of links as described above with simply annotating links as visited (yellow ball) and not visited (orange ball). This second type of annotation used in the control group is comparable to the usual annotation performed by WWW browsers that annotate links that have already been visited and that are cached. The second treatment contrasted providing a NEXT best step button with a version without this navigation button. The results of this study showed some hint of how these adaptive techniques may influence the learning process. Subjects who had no previous experience with any programming language tried to learn longer with ELM-ART when they were guided by the system using a NEXT button. This can be easily interpreted when one looks at the navigation behavior of the complete beginners more closely. All but one of the beginners had no experience in using a WWW browser. That is, these subjects profited from being guided directly by the system when using the NEXT button. Without such a button, they had to navigate through the course materials on their own. Learning to navigate through hypertext in addition to learning the programming language may have been too difficult. So individual adaptive guidance by the system is especially helpful for the complete beginners. In turn, most subjects who were already familiar with at least one other programming language (and also were familiar with Web browsers) were not affected by adaptive guidance. They were more pleased with the link annotation and stayed with the learning system longer when links were annotated adaptively. These results suggest that different adaptation technologies may be relevant to the users with different starting knowledge level of the subject. Similar results on the connection between starting level of knowledge and relevant adaptation techniques were presented in (Specht & Kobsa, 1999).

The hypothesis in the second study postulates that both adaptive navigation support and individual curriculum sequencing with the NEXT button reduce the number of navigation steps. Both techniques should have an additive effect on the navigation process. Data seem to support the hypothesis only partially. Subjects that are individually guided by using a NEXT button needed fewer steps to finish the first lesson than subjects without such an option did. The adaptive link annotation does not seem to have any systematic effect on the number of navigation steps. These very small effects observed from the first lesson fade away during the following lessons. That is, only in the beginning does individual guidance by the system help learners to follow an optimal path through the curriculum. Later on, all subjects understand the simple hierarchical architecture of the programming course and most of them follow the best learning path without any guidance.

These results do not mean that adaptive link annotation and adaptive curriculum sequencing are not as important as expected. As could be shown in the data above, these techniques are especially useful in the starting phase when users, especially beginners, are often frustrated. And, these techniques will presumably be helpful to advanced users who already posses some of the to-be-learned knowledge. In this case, a system that is able to adapt to the particular user will be helpful in navigating him or her around all the pages that the system infers to be known. However, this has to be shown in a different study.

Results of Learning with ELM-ART

Crucial to a learning situation is how successful and how fast learners complete the course. As ELM-ART directly stems from the on-site learning environment ELM-PE (Weber &

Möllenberg, 1995), comparing results of learning with ELM-ART to previous results from learning with ELM-PE will give an idea of how well one can learn with a web-based learning system. We compared 28 subjects learning with ELM-PE with 23 subjects using ELM-ART. In each case, subjects completed the first six lessons of our introductory LISP-course at the University of Trier and worked at three final programming tasks: A simple cdr-end-recursion A-LIST-TEST (Problem 1), a typical tree-recursion COUNT-ITEM (Problem 2), and a cdr-recursion LIST-UP-TO-ATOM (Problem 3). The third task was more difficult than the other ones because an atypical type of terminal case had to be coded. In these final programming tasks, learners could not use the intelligent diagnosis of the program code and there was no indication (e.g., by annotation) of whether the solution was correct or not. Therefore, learners had to test the programs on their own. The number of correct solutions to the programming tasks is a direct measure of whether users of the different systems were able to learn to program recursive functions in LISP.

Table 1 shows a comparison of the results of both groups learning with ELM-PE and ELM-ART, respectively. It has to be remarked that most of the subjects are students in psychology at the University of Trier and that the programming course is optional. That is, students are not required to attend this course. One can see (Table 1) that results in Problem 1 and Problem 2 are similar in both groups. This may be a ceiling effect. E.g., in the ELM-ART group only one subject did not solve the problems. The more difficult Problem 3 was solved more often in the ELM-ART group than in the ELM-PE group.

Table 1. Percentage of correct solution of the three final programming problems after lesson 6 in ELM-PE and ELM-ART.

Courses	**Problem 1** (A-List-Test)	**Problem 2** (Count-Item)	**Problem 3** (List-up-to-Atom)
ELM-PE (*N*=28)	100 %	93 %	54 %
ELM-ART (*N*=23)	96 %	96 %	87 %

It may be argued that learners with more previous experiences profit more from intelligent support systems than learners without previous knowledge. Therefore, we looked more carefully at the results of Problem 3 and distinguished learners with previous knowledge in other programming languages from learners without knowledge in any programming language (Table 2).

Table 2. Percentage of correct solution of the third final programming task with respect to previous programming knowledge.

Problem 3	**Previous Programming Knowledge**	
Courses	**with**	**without**
ELM-PE	64.4 % (*N*=14)	42.9 % (*N*=14)
ELM-ART	91.7 % (*N*=12)	81.8 % (*N*=11)

Results of this comparison show that learners without previous programming knowledge profited more from the learning system in the ELM-ART group than in the ELM-PE group ($F(1,47)=7.09$, $p=0.011$). This finding is supported by feedback from users stressing the ease of use and of learning with ELM-ART.

There is one more finding that supports the success of ELM-ART. Students participating in our introductory LISP course at the University of Trier completed the first six lessons more quickly. More than two thirds of all students working with ELM-ART completed the first six lessons within six weeks or less, while more than two thirds of all students working with ELM-PE needed more than seven weeks. The main reason for this remarkable difference is the

availability of computers. The on-site learning environment ELM-PE was available only on specially equipped Macintosh computers in our department. These computers were reserved two hours a week for each student. Conversely, ELM-ART could be accessed from most computers in the university (and even from home). We observed students starting work in the morning before classes started, continuing during the lunch hour, and proceeding in the evening. Therefore, a lot of students finished the course within a very short period of time. This may be one reason for the very good results in the final programming tasks. Students were able to concentrate on learning LISP much more than with the more occasional learning and practice sessions required by ELM-PE.

Feedback from users learning with ELM-ART

ELM-ART now is running in the WWW for several years with hundreds of users from all over the world learning the first steps of programming in LISP with ELM-ART as well in German as in English. After finishing the course, many of the learners filled in an optional feedback questionnaire and submitted it. The overall results are very good and in many cases even enthusiastic. Almost all learners answered affirmatively that they suggested ELM-ART to other people to learn with the system. They confirmed that it was easy to navigate in ELM-ART and that working with the system in the Internet was fast enough (even from the opposite side of the world). The interactive LISP-evaluator was of much help. The diagnosis was helpful in most cases though in some cases the diagnosis failed to present understandable feedback. The adaptive examples were used not very often, however, in case it examples were used they turned out to be very helpful. Many users criticized that no parentheses matcher was offered to support coding programs. Most users wished to get more lessons and to have a similar system for other programming languages and many users did not need the frame on the right side of the window, especially they did not need the field to note individual remarks.

To sum up this summary on the feedback messages the following statements show the power of an adaptive versatile learning system in the WWW: " *It was great fun for me*", " *Good, concise and useful*", "*Hot stuff*", "*Finally, I understand how this programming language functions. Our teacher missed it up to now in the lectures* (translated from German)", " *I think this is an very efficient tutorial, I think I am able to grasp the main idea of programming in LISP in a relative short time*", " *This is an excellent example of how software for teaching purposes should work*", " *I was very impressed with this tutorial. It provided the equivalent of almost a full course with teacher help which was still quickly completed and not at all frustrating*", " *This is one of the best web sites I've ever seen. I've been trying to learn LISP for quite a number of years, and have never been able to grok it. This seems unusual, since I had written several compilers and interpreters for the FORTH programming language (kind of the anti-LISP, where everything is post-fix, rather than pre-fix)*".

Discussion

The main result of the evaluation studies already done with ELM-ART and reported with this paper is the finding that Web-based educational systems can be as effective as traditional elaborated ITS. At the same time, the results of our investigations on the adaptive features of ELM-ART are ambiguous. Users with some previous knowledge (from another programming language and from browsing the Web) profited from link annotation, very often looked around in the course and then proceeded with learning. The more a user already knew with respect to the new domain and to the type of learning environment, the more this user will profit from the freedom to adapt the system to his or her needs and to explore the new environment (where annotation and other forms of adaptivity can be of much help). On the contrary, novice programmers (mostly without extensive experience in Web-browsing) more often followed the ideal learning path pointed out by the system and used the system‘s guidance. Therefore, they were not so much influenced by the other adaptive features (like link annotation).

These effects faded away the longer users worked with the course. A more general consequence of these results is that adaptive annotation and individual guidance is especially helpful and effective when starting in a new environment. This leads to the assumption that

these adaptive features will be especially useful in continuing education, where users start with at least some previous knowledge and the system can help the learner to concentrate on those sections and units of the domain in question that are new to the learner. However, these effects have not been addressed in the studies reported in this paper. There is an empirical study by Specht and Weber (1997) showing that users retaking a statistics course profited from adaptive features of the learning system. Learners had to fulfill an introductory questionnaire to show what they already knew. The group that was guided by the system to only those parts of the course that they had to repeat profited more from the repetition course than learners who had to read and work at all pages without adaptive guidance.

It remains an open question how adaptive systems can be tested and evaluated effectively. A lot of first investigations on this topic either was not able to show any effects or only had small or ambiguous effects. Especially in complex systems (like ELM-ART), adaptivity has to compete with other features designed to aid the user or learner. Therefore, it is difficult to partial out the effects of one single feature. On the other hand, more controlled psychological experiments, that concentrate on one or two single effects within a specially designed experimental environment may not be ecologically valid. We hope that in the future a methodological paradigm can be found that offers both controlled and ecologically valid experiments.

LESSONS LEARNED: FROM ITS TO WEB-BASED EDUCATION

The system, ELM-ART, described in this paper is an example of how an ITS can be re-considered and redesigned when being converted from a standalone to a Web-based system. Instead of developing a Web-based "copy" of its ancestor system ELM-PE (the way chosen by several research teams) we have tried to make ELM-PE benefit as much as possible from being a Web-based system and to integrated it into the context of Web-based education (Table 3). Since the Web is a natural platform for developing hypertext learning materials, we have decided to extend the original system with an electronic textbook component, added a hyperspace of functions and examples, and provided adaptive navigation support. Since students in a Web-based education context have to be able to communicate with each other and with the teachers, we have provided several ways of communication and collaboration support. Since the variety of students who can access the system on the Web is much larger than in a homogeneous university classroom, we have added an open and editable student model and comprehensive customization. To avoid the problem of installing LISP environments on different computers used by "Web students" we have provided a LISP interpreter with advanced visualization as a part of the system. The resulting system integrates the features of courseware management systems, electronic textbooks, learning environments, and intelligent tutoring systems.

ELM-ART clearly shows a number of benefits provided by a Web-based system. From the developer's prospect, the Web nature of the system makes the job of maintaining and enhancing the system much easier. First, the Web context provides a seamless connection between users and developers. With indirect feedback (log analysis) and direct feedback from the users the system can be improved permanently. This is the experience we had in developing ELM-ART over a long period of time. A lot of features were initiated by suggestions from users and the content of the course has been improved by getting feedback on a lot of shortcomings in the first versions. These revisions easily could be made available to all users because ELM-ART, as a Web-based learning environment, is a centralized learning system implemented on one server only. Therefore, changing the system on the server directly affected all learners. With on-site systems, new versions would have to be installed on each site, and with CD-ROM-based learning systems, distributing a new version is much more complicated and expensive.

From the student's prospect, the system provides a better support in the process of learning. To fit the original ELM-PE system to the Web context (for example, to fit several student models to the memory space of a single server-based system) we have to reduce the "intelligence" of some features. However, the comparison of ELM-ART and ELM-PE described in section 10.2 demonstrated the ability to learn "anyplace anywhere" and the

integrated textbook more than compensate for the decrease in "intelligence" - the students' performance with ELM-ART is better than with ELM-PE.

Table 3. Feature-by-feature comparison of Web-based systems used for teaching programming-related subjects. **Bold** font is used to stress features implemented with adaptive and intelligent technologies. *Italicised* font is used to stress other advanced technologies. The table shows that typical Courseware Management systems (CMS) such as Blackboard CourseInfo (Blackboard, 2000) or WebCT (WebCT, 1999) can support almost all basic features though on a very primitive level. Existing adaptive and intelligent Web-based systems described in Brusilovsky (1999) and some other advanced Web-based systems can support the whole range of features, though typical systems support one feature only. ELM-ART can support all key features while supporting the majority of them on the currently most advanced level.

	Typical CMS	ELM-ART	Other advanced Web-based systems
Hierarchically organized course material	Non-adaptive	**Adaptive link annotation and direct guidance**	**InterBook** (Brusilovsky et al., 1998)**, KBS-Hyperbook** (Henze et al., 1999)
Auto-evaluated questions	Non-adaptive	**Adaptive**	**SIETTE** (Rios et al., 1999)
Executable program examples	Non-adaptive, not-executable: example code only	**Adaptive example recommendation** *Non-adaptive visual execution*	*Non-adaptive visual execution: JELIOT* (Haajanen et al., 1997), *ISVL* (Domingue & Mulholland, 1998)
Programming problems to be submitted and evaluated	Program submission only (to be evaluated by a human tutor)	**Intelligent diagnosis** *Test-based automatic evaluation*	**Intelligent diagnosis: VC Prolog Tutor** (Peylo et al., 2000) *Test-based automatic evaluation: Web-Ceilidh:* (Foxley et al., 1998)
Student portfolio: analysed examples and own solutions	Could be accessed, but no special convenient tools	**Knowledge-based access, adaptive navigation support**	**Knowledge-based access: KBS-Hyperbook** (Henze & Nejdl, 2001)
Communication and collaboration	Announcements, e-mail, chat, forums	E-mail, chat, forums	**Adaptive peer help: IHD** (Greer et al., 1998)**,** *Debugging chat:*(Domingue & Mulholland, 1998)
Course monitoring	Grades and page access	*Concept level progress*	**Adaptive monitoring: HyperClassroom** (Oda, Satoh & Watanabe, 1998)

Overall, our experience of developing and using ELM-ART shows the need to move from small ITS supporting limited set of functionalities to versatile learning environments suitable for practical Web-based education. Unlike many others Web-based ITS that were never being used in real Web-based education for more than a trial period, ELM-ART was able to survive successfully several years of practical use in teaching a real university course over the Web. It was not due to the fact that every feature in ELM-ART was implemented on the most advance level or with the use of AI techniques. As we see from Table 3, it is true only for a subset of features. Other known systems, for example, can support collaboration or course monitoring better than ELM-ART. But it was in large extent due to the fact that in ELM-ART we have tried to provide support for as many needs of a student and a teacher as possible. Users do appreciate comprehensive versatile system. The recent competition between specialized and integrated course management systems in Web-based education context (see Introduction) has demonstrated this fact again.

We think that the Web can greatly benefit ITS (as a research field) and can help ITS to move from laboratories (where most of these "intelligent" system are used, due to the enormous requirements in computing power and capacity) to classrooms and to permanent availability in Web-based learning. However, we also think that on the way to the practical Web-based education existing ITS has to migrate from "single weapon" systems to comprehensive versatile environments. We have tried to start this process in 1996 with the first version of ELM-ART. More recently we were very pleased to see other emerging versatile AIWBES that provide on-

line learning material, support interactive problem solving and evaluate student's progress (Heift & Nicholson, 2001; Melis et al., 2001; Peylo et al., 2000). We hope that together with these systems, ELM-ART that stands between ITS and commercial CMS (Table 5) can provide a convincing example for the researchers and developers who works on advanced systems for Web-based instruction.

Acknowledgments

Part of this work is supported by a Grant from "Stiftung Rheinland-Pfalz für Innovation" grant to the first author and by fellowships from "Alexander von Humboldt-Stiftung" and James S. McDonnell Foundation to the second author.

References

Anderson, J. R., Conrad, F. G., and Corbett, A. T. (1989) Skill acquisition and the LISP Tutor. *Cognitive Science* 13, 467-505.

Barr, A., Beard, M., and Atkinson, R. C. (1976) The computer as tutorial laboratory: the Stanford BIP project. *International Journal on the Man-Machine Studies* 8 (5), 567-596.

Blackboard (2000) CourseInfo 3.0, Blackboard Inc. http://company.blackboard.com/CourseInfo/index.html (Accessed 25 December, 2000)

Bontcheva, K. (2001) Tailoring the content of dynamically generated explanations. In: M. Bauer, P. J. Gmytrasiewicz and J. Vassileva (eds.) *User Modeling 2001*. Lecture Notes on Artificial Intelligence, Vol. 2109, (Proceedings of 8th International Conference on User Modeling, UM 2001, Sonthofen, Germany, July 13-17, 2001) Berlin: Springer-Verlag, pp. 213-215.

Brusilovsky, P. (1994) Explanatory visualization in an educational programming environment: connecting examples with general knowledge. In: B. Blumenthal, J. Gornostaev and C. Unger (eds.) *Human-Computer Interaction*. Lecture Notes in Computer Science, Vol. 876, (Proceedings of 4th International Conference on Human-Computer Interaction, EWHCI'94, St. Petersburg, Russia, August 2-5, 1994) Berlin: Springer-Verlag, pp. 202-212.

Brusilovsky, P. (1995) Intelligent tutoring systems for World-Wide Web. In: R. Holzapfel (ed.) Proceedings of Third International WWW Conference, Darmstadt, Darmstadt, April 10-14, 1995, Fraunhofer Institute for Computer Graphics, pp. 42-45, also available at http://www.igd.fhg.de/archive/1995_www95/proceedings/posters/48/index.html.

Brusilovsky, P. (1999) Adaptive and Intelligent Technologies for Web-based Education. In C. Rollinger and C. Peylo (eds.), *Künstliche Intelligenz* (4), Special Issue on Intelligent Systems and Teleteaching, 19-25, http://www2.sis.pitt.edu/~peterb/papers/KI-review.html.

Brusilovsky, P. and Eklund, J. (1998) A study of user-model based link annotation in educational hypermedia. In P. Carlson (ed.) *Journal of Universal Computer Science* 4 (4), Special Issue on Assessment Issues for Educational Software, 429-448, http://www.iicm.edu/jucs_4_4/a_study_of_user.

Brusilovsky, P., Eklund, J., and Schwarz, E. (1998) Web-based education for all: A tool for developing adaptive courseware. *Computer Networks and ISDN Systems* (Proceedings of Seventh International World Wide Web Conference, 14-18 April 1998) 30 (1-7), 291-300.

Brusilovsky, P. and Miller, P. (2001) Course Delivery Systems for the Virtual University. In: T. Tschang and T. Della Senta (eds.): *Access to Knowledge: New Information Technologies and the Emergence of the Virtual University*. Amsterdam: Elsevier Science, pp. 167-206.

Brusilovsky, P. and Pesin, L. (1998) Adaptive navigation support in educational hypermedia: An evaluation of the ISIS-Tutor. *Journal of Computing and Information Technology* 6 (1), 27-38.

Brusilovsky, P., Schwarz, E., and Weber, G. (1996a) ELM-ART: An intelligent tutoring system on World Wide Web. In: C. Frasson, G. Gauthier and A. Lesgold (eds.) *Intelligent Tutoring Systems*. Lecture Notes in Computer Science, Vol. 1086, (Proceedings of Third International Conference on Intelligent Tutoring Systems, ITS-96, Montreal, June 12-14, 1996) Berlin: Springer Verlag, pp. 261-269.

Brusilovsky, P., Schwarz, E., and Weber, G. (1996b) A tool for developing adaptive electronic textbooks on WWW. In: H. Maurer (ed.) *Proceedings of WebNet'96*, World Conference of the Web Society, San Francisco, CA, October 15-19, 1996, AACE, pp. 64-69, also available at http://www.contrib.andrew.cmu.edu/~plb/WebNet96.html.

Brusilovsky, P. and Weber, G. (1996) Collaborative example selection in an intelligent example-based programming environment. In: D. C. Edelson and E. A. Domeshek (eds.) *Proceedings of International Conference on Learning Sciences*, ICLS'96, Evanston, IL, USA, AACE, pp. 357-362, also available at http://www2.sis.pitt.edu/~peterb/papers/icls96.html.

Bull, S. and Pain, H. (1995) "Did I say what I think I said, and do you agree with me?" Inspecting and questioning the student model. In: J. Greer (ed.) *Proceedings of AI-ED'95*, 7th World Conference on Artificial Intelligence in Education, Washington, DC, 16-19 August 1995, AACE, pp. 501-508.

Burow, R. and Weber, G. (1996) Example explanation in learning environments. In: C. Frasson, G. Gauthier and A. Lesgold (eds.) *Intelligent Tutoring Systems*. Lecture Notes in Computer Science, Vol. 1086, (Proceedings of Third International Conference on Intelligent Tutoring Systems, ITS '96, Montréal, June 12-14) Berlin: Springer-Verlag, pp. 457-465.

Corbett, A. T. and Anderson, J. A. (1992) Knowledge tracing in the ACT programming tutor. In: *Proceedings of 14-th Annual Conference of the Cognitive Science Society*.

Domingue, J. and Mulholland, P. (1998) An Effective Web Based Software Visualization Learning Environment. *Journal of Visual Languages and Computing* 9 (5), 485-508.

Eklund, J. (1996) Knowledge-based navigation support in hypermedia courseware using WEST. *Australian Educational Computing* 11 (2), 10-14.

Foxley, E., Benford, S., Burke, E., and CEILLIDH Development Team (1998) WebCeilidh, University of Nottingham. http://www.cs.nott.ac.uk/~ceilidh (Accessed 28 July, 1998)

Greer, J., McCalla, G., Cooke, J., Collins, J., Kumar, V., Bishop, A., and Vassileva, J. (1998) The intelligent helpdesk: Supporting peer-help in a university course. In: B. P. Goettl, H. M. Halff, C. L. Redfield and V. J. Shute (eds.) *Intelligent Tutoring Systems*. Lecture Notes in Computer Science, Vol. 1452, (Proceedings of 4th International Conference, ITS-98, San Antonio, TX, August 16-19, 1998) Berlin: Springer-Verlag, pp. 494-503.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., and Vanninen, P. (1997) Animation of user algorithms on the Web. In: Proceedings of VL '97, IEEE Symposium on Visual Languages, , IEEE, pp. 360-367, also available at http://www.cs.helsinki.fi/research/aaps/Jeliot/vl.ps.gz.

Heift, T. and Nicholson, D. (2001) Web delivery of adaptive and interactive language tutoring. *International Journal of Artificial Intelligence in Education* 12 (this issue).

Henze, N., Naceur, K., Nejdl, W., and Wolpers, M. (1999) Adaptive hyperbooks for constructivist teaching. In C. Rollinger and C. Peylo (eds.), *Künstliche Intelligenz* (4), 26-31.

Henze, N. and Nejdl, W. (2001) Adaptation in open corpus hypermedia. *International Journal of Artificial Intelligence in Education* 12 (this issue).

Hoppe, U. (1995) Use of multiple student modeling to parametrize group learning. In: J. Greer (ed.) *Proceedings of AI-ED'95*, 7th World Conference on Artificial Intelligence in Education, Washington, DC, 16-19 August 1995, AACE, pp. 234-249.

Kay, J. (1995) The UM toolkit for cooperative user models. *User Modeling and User-Adapted Interaction* 4 (3), 149-196.

Kolodner, J. L. (1993) Case-based reasoning. San Mateo, CA: Morgan Kaufmann.

Mallery, J. C. (1994) A Common LISP hypermedia server. In: Proceedings of the First International Conference on the World-Wide Web, May 25, 1994.

Melis, E., Andres, E., Büdenberder, J., Frishauf, A., Goguadse, G., Libbrecht, P., Pollet, M., and Ullrich, C. (2001) A generic and adaptive Web-based learning environment. *International Journal of Artificial Intelligence in Education* 12 (this issue).

Milosavljevic, M. and Oberlander, J. (1998) Dynamic hypertext catalogues: Helping users to help themselves. In: K. Grønbæk, E. Mylonas and F. M. Shipman III (eds.) *Proceedings of*

Ninth ACM International Hypertext Conference (Hypertext'98), Pittsburgh, USA, June 20-24, 1998, ACM Press, pp. 123-131.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986) Explanation-based generalization: a unifying view. *Machine Learning* 1 (1), 47-80.

Morelos-Borja, H. (1998) Effects of Automatic Assignment of Peers to Tutor/Tutee Roles in Web-based Distance Education, Report, Report, School of Computer Science, University of Central Florida, http://www.cs.ucf.edu/~morelos/papers/Fall98TechReport.rtf.

Nakabayashi, K., Koike, Y., Maruyama, M., Touhei, H., Ishiuchi, S., and Fukuhara, Y. (1995) An intelligent tutoring system on World-Wide Web: Towards an integrated learning environment on a distributed hypermedia. In: H. Maurer (ed.) *Proceedings of ED-MEDIA'95* - World conference on educational multimedia and hypermedia, Graz, Austria, June 17-21, 1995, AACE, pp. 488-493.

Oda, T., Satoh, H., and Watanabe, S. (1998) Searching deadlocked Web learners by measuring similarity of learning activities. In: *Proceedings of Workshop "WWW-Based Tutoring" at 4th International Conference on Intelligent Tutoring Systems (ITS'98)*, San Antonio, TX, August 16-19, 1998, also available at http://www.sw.cas.uec.ac.jp/~watanabe/conference/its98workshop1.ps.

Oppermann, R. (1994) Adaptively supported adaptability. *International Journal on Human-Computer Studies* 40, 455-472.

Peylo, C., Thelen, T., Rollinger, C., and Gust, H. (2000) A Web-based intelligent educational system for PROLOG. In: *Proceedings of Workshop on Adaptive and Intelligent Web-based Education Systems at 5th International Conference on Intelligent Tutoring Systems (ITS'2000)*, Montreal, Canada, June 19, 2000 Published as Report,, Institute for Semantic Information Processing, University of Osnabrück, Osnabrück. pp. 85-96.

Rios, A., Millán, E., Trella, M., J.L., P., and Conejo, R. (1999) Internet based evaluation system. In: S. P. Laojie and M. and Vivet (eds.) *Artificial Intelligence in Education: Open Learning Environments*. (Proceedings of AI-ED'99, Le Mans, France) Amsterdam: IOS Press, pp. 387-394.

Schöch, V., Specht, M., and Weber, G. (1998) "ADI" - an empirical evaluation of a tutorial agent. In: T. Ottmann and I. Tomek (eds.) *Proceedings of ED-MEDIA/ED-TELECOM'98* - 10th World Conference on Educational Multimedia and Hypermedia and World Conference on Educational Telecommunications, Freiburg, Germany, June, 20-25, 1998, AACE, pp. 1242-1247.

Schwarz, E., Brusilovsky, P., and Weber, G. (1996) World-wide intelligent textbooks. In: *Proceedings of ED-TELECOM'96* - World Conference on Educational Telecommunications, Boston, MA, June 17-22, 1996, AACE, pp. 302-307, also available at http://www.contrib.andrew.cmu.edu/~plb/ED-MEDIA-96.html.

Self, J. (1995) Dormorbile: a vehicle for metacognition. In: T. W. Chan and J. A. Self (eds.): *Emerging Computer Technologies in Education*. Charlottesville: AACE.

Specht, M. and Kobsa, A. (1999) Interaction of domain expertise and interface design in adaptive educational hypermedia. In: P. Brusilovsky and P. D. Bra (eds.) *Proceedings of Second Workshop on Adaptive Systems and User Modeling on the World Wide Web*, Toronto and Banff, Canada, May 11 and June 23-24, 1999 Published as Computer Science Report, No. 99-07, Eindhoven University of Technology, Eindhoven. pp. 89-93.

Specht, M. and Oppermann, R. (1998) ACE - Adaptive Courseware Environment. In P. Brusilovsky and M. Milosavljevic (eds.), *The New Review of Hypermedia and Multimedia* 4, Special Issue on Adaptivity and user modeling in hypermedia systems, 141-161.

Specht, M. and Weber, G. (1997) Kognitive Lernermodellierung. *Kognitionswissenschaft* 6 (4), 165-176.

Specht, M., Weber, G., Heitmeyer, S., and Schöch, V. (1997) AST: Adaptive WWW-Courseware for Statistics. In: P. Brusilovsky, J. Fink and J. Kay (eds.) *Proceedings of Workshop "Adaptive Systems and User Modeling on the World Wide Web"* at 6th International Conference on User Modeling, UM97, Chia Laguna, Sardinia, Italy, June 2, 1997, pp. 91-95, also available at http://www.contrib.andrew.cmu.edu/~plb/UM97_workshop/Specht.html.

Tudhope, D., Taylor, C., and Benyon-Davies, P. (1995) Navigation via similarity in hypermedia and information retrieval. In: R. Kuhlen and M. Ritterberg (eds.) *Proceedings of HIM'95*, Konstanz, Konstanz, April 1995, Universitätverlag Konstanz, pp. 203-218.

WebCT (1999) World Wide Web Course Tools 1.3.1, WebCT Educational Technologies, Vancouver, Canada. http://www.webct.com (Accessed 15 February, 1999)

Weber, G. (1996a) Episodic learner modeling. *Cognitive Science* 20 (2), 195-236.

Weber, G. (1996b) Individual selection of examples in an intelligent learning environment. *Journal of Artificial Intelligence in Education* 7 (1), 3-31.

Weber, G. (1999) Adaptive learning systems in the World Wide Web. In: J. Kay (ed.) *Proceedings of 7th International Conference on User Modeling*, UM99, Banff, Canada, June 20-24, 1999, Wien: SpringerWienNewYork, pp. 371-378.

Weber, G., Kuhl, H.-C., and Weibelzahl, S. (2001) Developing adaptive internet based courses with the authoring system NetCoach. In: P. D. Bra, P. Brusilovsky and A. Kobsa (eds.) *Proceedings of Third workshop on Adaptive Hypertext and Hypermedia*, Sonthofen, Germany, July 14, 2001, Technical University Eindhoven, pp. 35-48, also available at http://wwwis.win.tue.nl/ah2001/papers/GWeber-UM01.pdf.

Weber, G. and Möllenberg, A. (1995) ELM-Programming-Environment: A Tutoring System for LISP Beginners. In: K. F. Wender, F. Schmalhofer and H.-D. Böcker (eds.): *Cognition and Computer Programming*. Norwood, NJ: Ablex, pp. 373-408.

Weber, G. and Specht, M. (1997) User modeling and adaptive navigation support in WWW-based tutoring systems. In: A. Jameson, C. Paris and C. Tasso (eds.): *User Modeling: Proceedings of the Sixth International Conference, UM97*. Vienna, New York: Springer, pp. 289-300.